2019-09

VISION-BASED TERRAIN CLASSIFICATION AND LEARNING TO IMPROVE AUTONOMOUS GROUND VEHICLE NAVIGATION IN OUTDOOR ENVIRONME

Lebrun, Caliph

Monterey, CA; Naval Postgraduate School

NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

VISION-BASED TERRAIN CLASSIFICATION AND LEARNING TO IMPROVE AUTONOMOUS GROUND VEHICLE NAVIGATION IN OUTDOOR ENVIRONMENTS

by

Caliph Lebrun

September 2019

Thesis Advisor: Xiaoping Yun
Second Reader: James Calusdian

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE	*Form Approved OMB No. 0704-0188*

1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** September 2019	**3. REPORT TYPE AND DATES COVERED** Master's thesis

4. TITLE AND SUBTITLE VISION-BASED TERRAIN CLASSIFICATION AND LEARNING TO IMPROVE AUTONOMOUS GROUND VEHICLE NAVIGATION IN OUTDOOR ENVIRONMENTS	**5. FUNDING NUMBERS**
6. AUTHOR(S) Caliph Lebrun	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000	**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A	**10. SPONSORING / MONITORING AGENCY REPORT NUMBER**

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release. Distribution is unlimited.	**12b. DISTRIBUTION CODE** A

13. ABSTRACT (maximum 200 words)

Terrain is an important factor for autonomous ground vehicles (AGV), potentially ruining a mission or the platform itself. The purpose of this thesis is to develop a method for an AGV to identify and avoid hazardous terrain. This work builds on a previously developed system that uses artificial potential fields to avoid obstacles and navigate to a goal. Terrain was identified by developing a random forest machine-learning algorithm, classifying terrain as hazardous or traversable. The random forest was grown using data from images collected during this work. The classification of hazardous terrain was used to generate a repulsive force for use with artificial potential fields. The system was designed to avoid known areas of hazardous terrain using path planning, developing paths using approximate cell decomposition and the A* search algorithm. Tests of the developed random forest revealed accurate classification capabilities for all terrain types, but a tendency to misclassify certain terrain types. Portions of the navigation solution were simulated and confirmed the path planning capability. Trials conducted in a real-world environment revealed the solution stopped the AGV from entering hazardous terrain, and successfully planned routes around hazardous terrain. Improvements to the localization solution will allow the AGV to perform more consistently and over longer ranges.

14. SUBJECT TERMS terrain classification, machine learning, random forest, unmanned ground robot, computer vision, autonomous, approximate cell decomposition, A* search, artificial intelligence	**15. NUMBER OF PAGES** 135
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	**18. SECURITY CLASSIFICATION OF THIS PAGE**	**19. SECURITY CLASSIFICATION OF ABSTRACT**	**20. LIMITATION OF ABSTRACT**
Unclassified	Unclassified	Unclassified	UU

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89) Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release. Distribution is unlimited.

VISION-BASED TERRAIN CLASSIFICATION AND LEARNING TO IMPROVE AUTONOMOUS GROUND VEHICLE NAVIGATION IN OUTDOOR ENVIRONMENTS

Caliph Lebrun
Captain, United States Marine Corps
BS, Mathematics, Eastern Michigan University, 2013
BS, Physics, Eastern Michigan University, 2013

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
September 2019

Approved by: Xiaoping Yun
Advisor

James Calusdian
Second Reader

Douglas J. Fouts
Chair, Department of Electrical and Computer Engineering

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

Terrain is an important factor for autonomous ground vehicles (AGV), potentially ruining a mission or the platform itself. The purpose of this thesis is to develop a method for an AGV to identify and avoid hazardous terrain. This work builds on a previously developed system that uses artificial potential fields to avoid obstacles and navigate to a goal. Terrain was identified by developing a random forest machine-learning algorithm, classifying terrain as hazardous or traversable. The random forest was grown using data from images collected during this work. The classification of hazardous terrain was used to generate a repulsive force for use with artificial potential fields. The system was designed to avoid known areas of hazardous terrain using path planning, developing paths using approximate cell decomposition and the A* search algorithm. Tests of the developed random forest revealed accurate classification capabilities for all terrain types, but a tendency to misclassify certain terrain types. Portions of the navigation solution were simulated and confirmed the path planning capability. Trials conducted in a real-world environment revealed the solution stopped the AGV from entering hazardous terrain, and successfully planned routes around hazardous terrain. Improvements to the localization solution will allow the AGV to perform more consistently and over longer ranges.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF ACRONYMS AND ABBREVIATIONS

AGV	autonomous ground vehicle
AI	artificial intelligence
GNSS	global navigation satellite system
GNSS/INS	GNSS-inertial navigation system
KML	keyhole markup language
LIDAR	light detection and ranging
MLS	minimum leaf size
P3-AT	pioneer 3-all terrain
ROI	region of interest
ROS	robot operating system
SSR	small scale representation
SURF	speeded up robust feature
USB	universal serial bus

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

Jesus, my rock and redeemer, thank you above all.

I would like to thank professors Brian Bingham, Roberto Cristi, Vladimir Dobrokhodov, Monique Fargues, and Doug Horner. Your instruction has helped me greatly throughout this work, and I have always looked forward to your presentation of complex and intriguing topics.

I would like to offer special thanks to my thesis advisors, Dr. Xiaoping Yun and Dr. James Calusdian. You have each shown me great patience and guidance throughout this process. Dr. Yun, thank you for your insight and tutelage through the research process. Dr. Calusdian, thank you for your assistance in framing the problem and your willingness to discuss concepts and issues encountered throughout the research.

Finally, I would like to thank my family. Kelycera, Linkon, Leia, and Malcolm, you are a driving force in my life, and I only hope you know how much each of you inspires me. To my wife, Lindsay, you are a better person than I am, and without you by my side I would not have been able to come this far. Thank you for standing with me, and for enduring many nights of my yammering on about robots and artificial intelligence, and the gobs of math scribbled on boards and mirrors.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

Two decades ago, daily interactions with artificial intelligence (AI) and robots was nonexistent outside of science fiction. Yet, today, people across the planet communicate with digital assistants empowered by AI to increase their productivity, and robotic vacuum cleaners careen through our homes reducing time spent on mundane tasks. This rapid development of technology in such a short period of time has not gone unnoticed by the Department of Defense; in 2018, a strategy governing artificial intelligence was released. The Summary of the 2018 Department of Defense Artificial Intelligence Strategy states, "we will use AI in a human-centered manner," implying the need for AI to aid—not replace—the user [1]. The 2018 Marine Corps Science and Technology Strategic Plan expands on this concept by focusing on "manned and unmanned teaming" as well as advanced robotics that support ground forces [2]. According to the Marine Corps S&T Strategy, the Marine Corps is seeking systems that "enable effective 'supervised autonomy' by a Marine user" and have features such as "teleoperation, machine vision, perception, obstacle avoidance, convoy following, and the ability to self-navigate preplanned routes" [2].

Each of the features highlighted from the Marine Corps S&T Strategy are captured in one of the most provocative areas of research and development of our time, self-driving technology. This technology seeks to harness AI for the safe and efficient transportation of people from point A to point B. In the 2018 Department of Defense Artificial Intelligence Strategy, AI is described as "the ability of machines to perform tasks that normally require human intelligence—for example recognizing patterns, learning from experience, drawing conclusions, making predictions, or taking action" [1]. In [3], Russell and Norvig address AI from the perspective of a rational agent. They describe a rational agent as one that takes the correct course of action with the information it has. Each of these perspectives applies to the fundamental problem of self-driving technology, and consequently the same technology the Marine Corps seeks to aid its warfighter.

A. MOTIVATION

Private industry and civilian universities are heavily invested in exploring self-driving vehicles. Many organizations have constrained their research to civilian applications that neglect the unique operating conditions pertinent to military applications. Military vehicles employing self-driving capabilities complete with autonomy, machine learning, and AI must be able to account for the austere environments and free-form nature of the routes they are able to execute. This research seeks to exploit the capabilities of AI and machine learning to equip an autonomous ground vehicle with the ability to avoid hazardous terrain encountered in unstructured environments.

Through the development of a single autonomous system that is capable of safe and effective navigation through its environment, further research may be conducted to leverage this technology for the warfighter. This could include expanding research into manned and unmanned infantry teaming applications or cooperative convoys of manned and unmanned systems. As the military industry continues to explore means to leverage AI, including autonomous/semi-autonomous systems and machine learning applications, it is important to conduct research with systems that use them to explore the capabilities, limitations, and areas for future research with a military perspective in mind.

B. PREVIOUS WORK

This thesis research falls under an umbrella project of the Naval Postgraduate School's Electrical and Computer Engineering Department's Control Systems and Robotics laboratory, wherein the desired end state is to develop a system capable of navigating from any location to another on the school campus. Prior thesis research investigated and developed an autonomous system capable of motion planning and user-assisted path planning. In [4], Calvin Hargadine developed and integrated the autonomous ground vehicle (AGV) that is modified for use in this thesis research. His work focused on building a robotic platform and implementing an algorithm for motion planning with a simple sensor suite. Matthew Audette expanded on this work in [5], wherein he developed a path-planning algorithm that used user-defined constraints to plan waypoints for the

AGV. The research conducted in this thesis addressed the inability of the existing system to safely or effectively travel over certain types of terrain.

Addressing this problem required developing a system that could identify and avoid problematic terrain and was also able to learn about its operating environment to plan routes to its goal. This work required developing a method of classifying terrain and developing a navigation solution that accounted for the information gathered from the workspace. There has been significant work in terrain classification, as discussed by Panagiotis Papadakis [6], wherein he categorized the research into two major categories: proprioceptive and exteroceptive data processing. Proprioceptive pertains to measuring information about the state of a robot, whereas exteroceptive relates to the world around a robot [7]. Papadakis further classified research involving exteroceptive data processing according to whether it is geometry- or appearance-based [6]. This thesis research focuses on determining terrain traversability by the appearance of the terrain. Khan et al. conducted the research most relevant to this work in [8]. Their work explores various machine-learning algorithms and image-based feature types to determine the combination that provided the highest accuracy in terrain classification. The work done in this thesis research differs from that done in [8] in that it highlights the constraints of processing time for near-real-time control.

The method used for developing the navigation solution borrows from the classic ideas presented by Jean Claude Latombe in [9], employing approximate cell decomposition, the A* search algorithm, and artificial potential fields. This requires developing a method of representing the workspace and then implementing a path planning algorithm for navigating to the goal site. This thesis work differs in that it operates on several levels to first refine the location of obstacles defined as points and then plan a route through the workspace.

C. PURPOSE AND GOAL

The purpose of this thesis was to develop a method of identifying and avoiding terrain that is hazardous to the operation of an autonomous ground vehicle. Two goals were set to achieve this purpose. The first of those goals was the development of a machine

learning algorithm that differentiates between terrain types with a high level of accuracy. The second goal was to develop a method for avoiding terrain identified as hazardous.

The machine-learning algorithm constrains this work to the Naval Postgraduate School campus. This is due to the nature of developing a machine-learning algorithm and the conditions of capturing training data. This does not prevent the algorithm from being viable in other locations and conditions; however, it is likely that, with changes to the environment, the performance of the machine-learning algorithm presented herein would differ. The equipment used along with the methods used in this research are discussed throughout the next five chapters. The hardware, software, and key concepts of the thesis work are discussed in Chapter II. The machine-learning algorithm developed for use in terrain classification, random forest, is discussed in Chapter III. Chapter IV contains the description of integrating the terrain classification into a navigation solution for avoiding hazardous terrain. The results of the laboratory tests, simulations, and experimentation conducted are detailed in Chapter V. Conclusions from the tests and the thesis work are discussed in Chapter VI, as are recommendations for future work.

II. SYSTEM DESIGN AND KEY CONCEPTS

To research and develop an artificially intelligent system capable of navigating around hazardous terrain requires a combination of hardware and software. The combination of hardware and software must allow the system to sense its environment, plan its actions, and act out those plans [7]. This work modifies the system developed in [4] and [5] to accomplish the desired goals. The hardware and software used in this thesis research are described in the first two sections of this chapter before several key concepts to this thesis work are addressed.

Key concepts of this thesis work are discussed in Section II.C to provide context to several of the topics used to achieve the thesis objectives. These include approximate cell decomposition, the A* search algorithm, and the random forest machine learning algorithm.

A. HARDWARE

The purpose of this thesis requires automating a robotic system for use in an outdoor environment. This requires a platform that is sturdy enough for outdoor performance, has the computational capability to automate the desired behavior, and a robust sensor package that allows the system to observe relevant elements of the environment.

1. Pioneer 3-AT

The Pioneer 3-AT (P3-AT) is a four-wheeled robotic platform developed by Adept MobileRobots, shown in Figure 1. It is capable of operations in indoor and outdoor environments [10]. It is described in [10] as being able to traverse sand, dirt, asphalt, and flooring with a maximum grade of 35%. The 5.4 cm clearance under the P3-AT bumpers restricts the platform from traversing terrain with significant dips and other variations [11]. Up to three 12-V batteries power the platform [10]. The P3-AT is capable of providing 5V or 12V to attached devices [10].

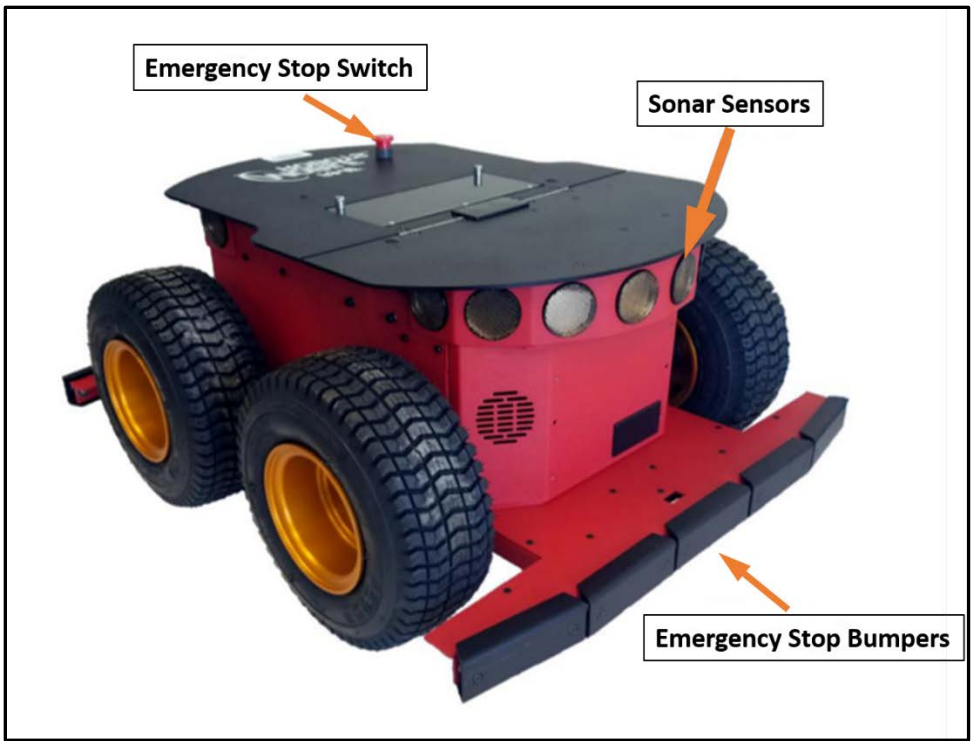


Figure 1. The P3-AT platform. Adapted from [4].

The P3-AT comes with a microcontroller that handles all lower level functions, such as the motor control, reporting wheel encoder data, and controlling the sonar sensors [11]. A serial connector allows interfacing the microcontroller with a mounted computer for developing higher level functionality [11]. The P3-AT can be driven with a joystick controller that is connected by a cable. As shown in Figure 1, the P3-AT comes with built-in sensors including sonar, emergency stop bumpers, and an emergency stop switch.

2. SlimPRO SP675P

Automating the P3-AT requires a mountable computational platform with a low power requirement. The SlimPRO SP675P, shown in Figure 2, meets these needs. It measures 5.75″ wide by 10.0″ long by 1.65″ tall and weighs approximately 2.4 kg [12]. The SlimPRO can be powered by the P3-AT as it only requires 60 W at 12 V [12]. It has four USB 3.0 ports and two USB 2.0 ports for sensor data reception [12]. The SlimPRO has a serial port that allows it to communicate with the microcontroller on the P3-AT [12]. This allows passing high level commands to the P3-AT, such as bearing and speed.

Figure 2. SlimPRO SP675P, front and back

3. Sensor Suite

a. Hokuyo UTM-30LX

The Hokuyo UTM-30LX, shown in Figure 3, is a two-dimensional light detection and ranging (LIDAR) system. It is capable of detecting objects at ranges between 0.1 and 30 meters [13]. It provides a 270° field of view with an angular resolution of 0.25° [13]. The Hokuyo provides scanning data to the SlimPRO over USB 2.0 [13]. Due to the 12 V voltage requirement of the Hokuyo, the P3-AT provides the necessary power for the system in this work [13].

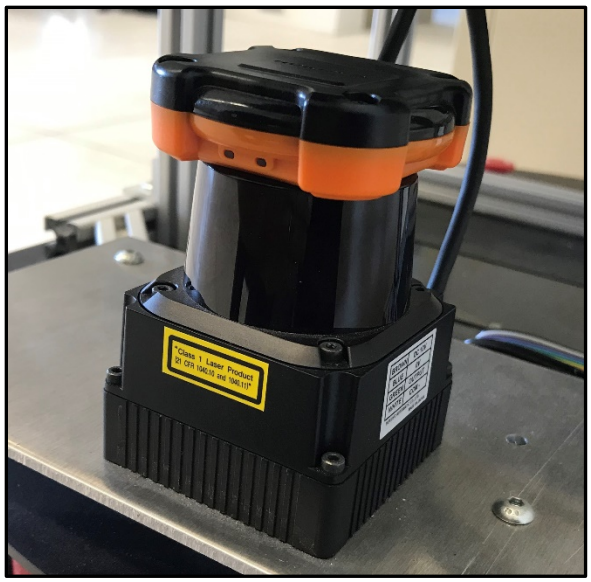


Figure 3. The Hokuyo UTM-30LX LIDAR system

b. LORD MicroStrain 3DM

Shown in Figure 4, the LORD MicroStrain 3DM-GX5-45 Global Navigation Satellite System-Aided Inertial Navigation System (GNSS/INS) provides a localization and orientation solution for the AGV. The GNSS/INS is described in [14] as providing position data accurate to within two meters in the horizontal plane and heading data that is accurate to within 0.8°. The system draws its power from the SlimPRO over USB 2.0 due to its low power requirement. It typically requires 700 mW, and it is able to operate on a DC power source with voltage ranging from 4 to 36 V [14]. The system also uses the USB 2.0 to communicate data to the SlimPRO [14]. The system can be calibrated to the environment in which it will be operated, accounting and adjusting for specific magnetic properties of that environment that may affect orientation data [15].

Figure 4. The LORD MicroStrain 3DM-GX5-45 GNSS/INS

c. P3-AT Chassis

As mentioned in Section II.A.1, the P3-AT chassis comes with several built-in sensors including an array of sonar sensors, emergency stop bumpers, and a manually operated emergency stop switch, as shown in Figure 1. This thesis work does not use the sonar sensors, as they provide the same capability as the LIDAR, but the LIDAR provides better resolution and range. There are ten emergency stop bumpers on the P3-AT chassis—five on the front and five on the back. These bumpers serve as a safety mechanism, temporarily stalling the chassis if something presses a bumper [11]. The manually operated emergency stop switch provides the user a hardware option for stopping the P3-AT. The system will not move while the switch remains activated.

d. Microsoft LifeCam HD-3000

The Microsoft LifeCam HD-3000, shown in Figure 5, is a commercial-off-the-shelf webcam that provides imagery to the SlimPRO via a USB 2.0 connection [16]. According to [16], the camera is capable of capturing “up to 30 frames per second”. Each image from the webcam has a resolution of 1280 by 800 pixels, and a 68.5°diagonal field of view [16]. Due to the intended use as a webcam the device has its focus fixed from 0.3 to 1.5 meters [16]. Consequently, any portions of the imaged area that are located outside of this range will not be in focus.

Figure 5. The Microsoft LifeCam HD-3000 USB webcam

B. SOFTWARE

The process of automating the robotic platform requires developing the ability for it to "sense, plan, and act" [7]. The hardware provides the sensing and acting capability, and the algorithms developed provide the planning capability. This thesis work makes extensive use of MATLAB for developing these algorithms. It has robust programming support and can be integrated with the Robot Operating System (ROS). ROS provides the backbone of the system, funneling information from sensors to algorithms and commands to actuators. Data analysis requires using MATLAB, ROS, and Google Earth Pro. Together they allow processing trial data and analyzing them on satellite imagery. Each of the software elements is described in this section.

1. MATLAB

For this work, the automation algorithms developed for the AGV were built in MATLAB 2018b. MATLAB is a programming environment designed for use by engineers and scientists, and is useful for algorithm development and data analysis [17]. MATLAB has many functions, toolboxes, and packages available for developing solutions to engineering problems [17]. This work makes use of the Computer Vision, Image Processing, Mapping, and Statistics and Machine Learning toolboxes along with the MATLAB Support Package for USB Webcams. Each of these is discussed briefly in this section.

The Computer Vision Toolbox is capable of providing the user with many computer vision specific algorithms and functions, streamlining the development of systems using computer vision [18]. This toolbox provides the ability to extract features from images throughout this work.

The Image Processing Toolbox enables “image processing, analysis, visualization, and algorithm development” [19]. The functions from this toolbox provide morphological algorithms for use in filtering and path planning, and prebuilt processes for finding the Hough transform.

The Mapping Toolbox provides functions for data export in file formats such as the keyhole markup language (KML) [20]. This enables analyzing geographic data from AGV interactions with the workspace.

In addition to the many statistical analysis functions available from the Statistics and Machine Learning Toolbox, the toolbox provides the ability to develop a wide variety of machine-learning algorithms [21]. This toolbox enabled development of a trained machine-learning model for terrain classification.

The MATLAB Support Package for USB Webcams provides a method of interfacing with the webcam via MATLAB commands and scripts [22].

2. Robot Operating System

ROS is a free software providing a framework for developing complex robotics software [23]. One of the features of ROS is the plumbing it provides by enabling message passing from sensors, algorithms, and actuators via its publish and subscribe process [24]. This streamlines the process of communicating sensor data to the algorithms used in automation. In this work, sensors publish data on the ROS network, and algorithms subscribe to the information they need. ROS also provides a recording feature that allows the user to record traffic communicated across the ROS network [24]. Data recorded in this way can be played back or used for further analysis [24].

3. Google Earth Pro

Google Earth Pro provides an Earth browser that uses the KML file format [25]. This allows visualizing information from AGV interactions in the world along with the routes it traveled. The specific release of Google Earth Pro used is 7.3.2.5776 (64-bit).

C. KEY CONCEPTS

Achieving the goals presented in Chapter I requires applying concepts from robotics and machine learning. Those concepts are discussed in this section to provide context prior to the application being discussed in Chapters III and IV.

1. Approximate Cell Decomposition

The goal of developing a method for avoiding terrain identified as hazardous led to the topic of path planning. Latombe discusses the problem of path planning in [9]. In his discussion on path planning, he details the need for a representation of the workspace. While he discusses several methods for doing this, this thesis borrows from the approximate cell decomposition method to develop a workspace representation for use in path planning. The application of approximate cell decomposition divides the free space into a collection of rectangular cells [9].

Dividing the workspace in this way allows obstacles to lie in several cells and partially occlude others. Paths will only be planned through open cells so mixed cells restrict the available routes to the goal. This is typically dealt with by recursively splitting mixed cells until the sub-cells are open or occluded [9]. It is practical to place a limit on the subdivision based on the kinematics of the robot being used. An example of a workspace segmented using the approximate cell decomposition method is shown in Figure 6. Open cells are white, mixed cells are white and gray checkered, and occluded cells are black. A graph search can use the decomposition to determine a path through the workspace. The graph search used in this thesis is the A* search method. It is described in the next section.

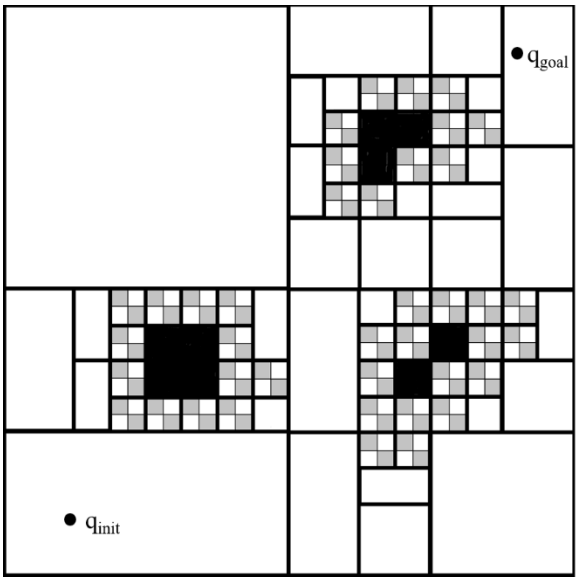


Figure 6. Example of approximate cell decomposition

2. A* Search

The A* (pronounced A star) search method is a classic search algorithm classified among the best-first searches [3]. When applied to a graph search, this algorithm finds the optimal route between two points or returns failure, if a route does not exist.

The A* algorithm uses a cost metric *f(q)* to find the optimal route to a goal [26]. According to the developers, Hart, Nilsson, and Raphael, the cost metric is comprised of two elements *g(q)* and *h(q)*, such that $f(q) = g(q) + h(q)$ [26]. The authors describe *g(q)* as the optimal cost to go from the initial node in the graph to the node *q*, and *h(q)* as the optimal cost between the node *q* and the goal. In application, a heuristic is used to estimate the cost. To ensure the path found is admissible, the heuristic cost estimate *h*(q)* must not overestimate the cost *h(q)* [26].

Starting with the initial node, the algorithm begins executing a recursive sequence. This sequence starts with assigning an estimated cost *f*(q)* to each node adjacent to the current node [26]. The algorithm then marks the current node as visited and checks for the unvisited node that it will visit next [9]. It visits the unvisited node with the cheapest estimated cost *f*(q)* [9]. This process continues until the algorithm finds the optimal route to the goal node or terminates by determining no route exists to the goal [26]. This thesis

modifies the application of approximate cell decomposition and A* search to account for the workspace representation and the obstacle avoidance constraints. These modifications are discussed in Sections IV.C.1.a and IV.C.1.b.

3. Random Forest

Random forest is a supervised machine-learning algorithm that results in an ensemble of bagged decision trees [27]. This means that each tree is grown from bootstrap aggregated portion of the training data [28]. According to Breiman in [27], developers can use random forests for regression or classification. This thesis work used random forests for classification and will address them from that perspective. Random forests are a group of decision trees grown under unique considerations. The description of growing a random forest begins with the process for growing a decision tree. After the process of growing a decision tree is covered, the unique methods employed for growing a random forest are discussed.

Decision trees are grown by recursively splitting a feature space, formed by the training data, until reaching the desired stopping condition [28]. This starts with the training set and defining a splitting method and stopping condition.

Training data consists of n observations where each observation is a feature vector $\boldsymbol{x}$ and a label y [29]. Feature vectors represent the information, called features, from the observation. These features either come from the raw data or are the result of some processing by the developer. The label y defines the category of the observation.

The splitting method determines where to split the feature space. This involves comparing potential child regions in the space. This thesis work makes use of an impurity-based splitting method that develops purer child regions from the parent regions. This means that each child region is proportionally more of one category than the other. This requires identifying the feature and value of the feature that provides the purest child regions after the split. When growing a decision tree the splitting criteria have access to all features from the training data [29].

The process of recursively splitting the regions based on feature values ends upon meeting the stopping condition [29]. Stopping conditions include, but are not limited to, reaching a maximum number of splits or the regions being pure [29].

To demonstrate the concept of a recursively split feature space, and how that translates to a decision tree, an example tree is grown here for illustration. The tree is generated from a benchmark data set used in statistics and machine learning, known as the Fisher iris data. This data consists of 150 observations of four features: sepal length and width and petal length and width [30]. It represents 50 observations of each of three species of iris: setosa, versicolor, and virginica [30].

This example uses two of the features from this data, sepal length and width, to grow the example decision tree. The observations of these features are plotted by species in Figure 7 to visualize the feature space.

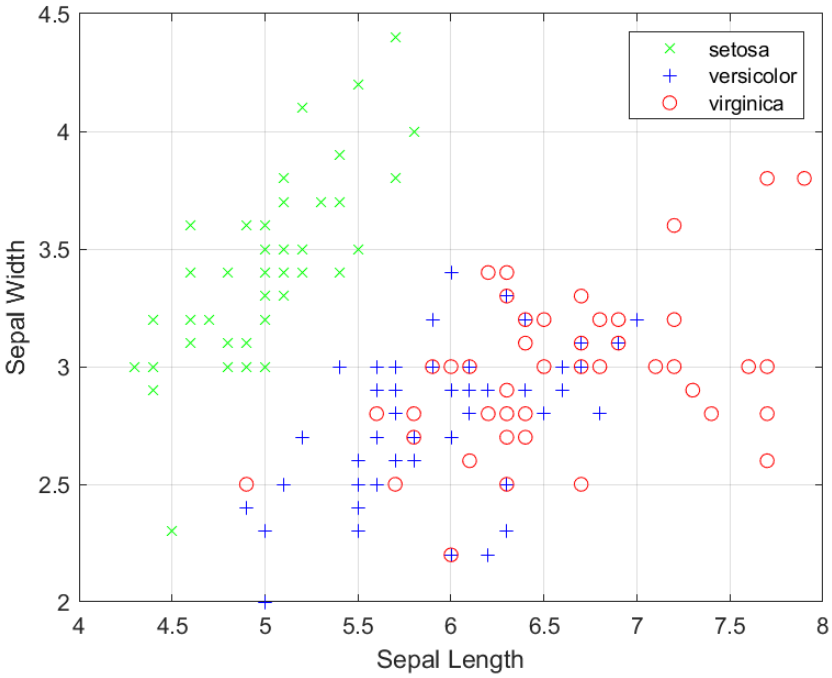


Figure 7. Fisher iris sepal data used to grow a decision tree. Adapted from [30].

A clear delineation between setosa and the other iris species is shown in Figure 7. The setosa data tends to be clustered in the upper left of the figure, whereas the versicolor and virginica tend toward the lower right. There are several instances of versicolor and virginica sharing space within the feature space. The division of the feature space determined through growing the decision tree in MATLAB is shown in Figure 8, and the corresponding tree is shown in Figure 9.

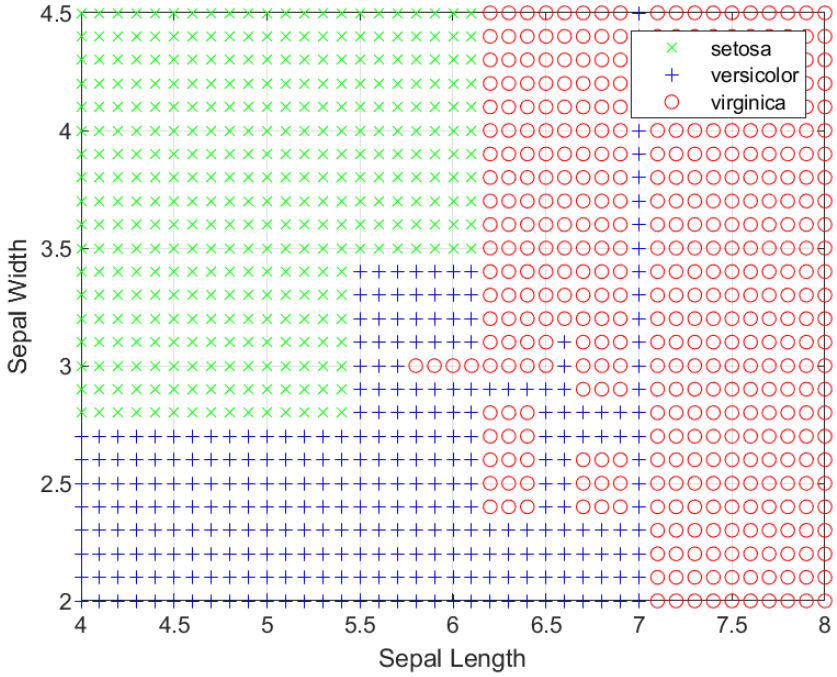


Figure 8. Feature space divided according to the decision tree grown for iris classification. Adapted from [30].

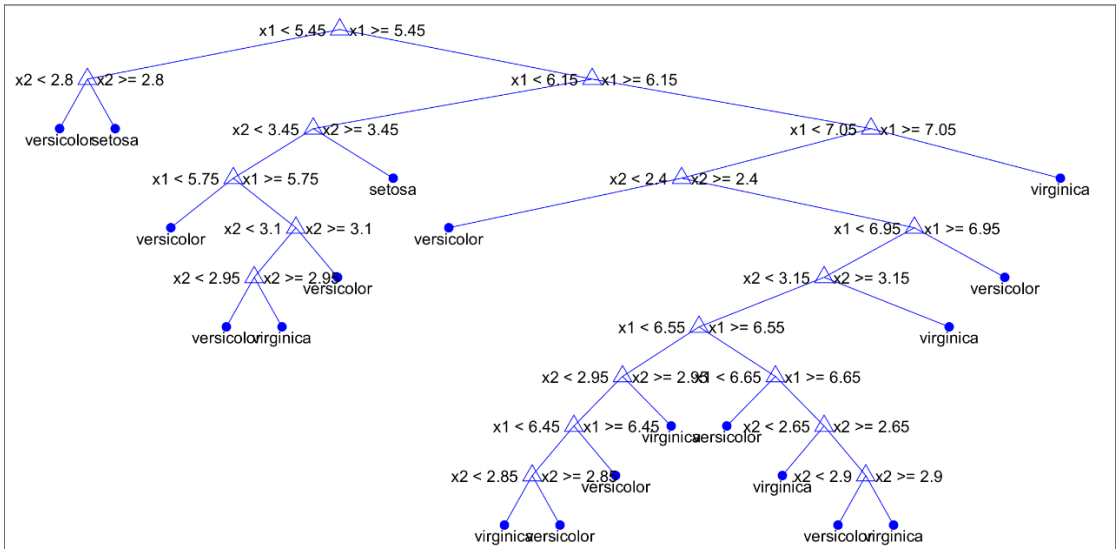


Figure 9. Decision tree grown from Fisher iris sepal data. Adapted from [30].

Figure 8 shows the division of the feature space defined by the sepal length and width according to the decision tree shown in Figure 9. In Figure 9, sepal length and width are annotated as x1 and x2, respectively. Comparing Figure 7 to Figure 8 shows the feature space matching quite well to the training data. One of the oddities is the portion of the space where the sepal length is equal to seven and all instances are assigned to the versicolor category regardless of sepal width. This is due to the greedy nature of decision trees. This greedy nature causes decision trees to overfit to the training data, and decision trees may not generalize well because of it [29]. There are several methods to address this, one of which is growing random forests [28]. As discussed, a random forest is an ensemble of decision tress grown under two special conditions.

The first condition addresses which observations are used to grow each tree. A random set of the observations are used for each tree [27]. Expanding on this concept, it is assumed that there are *n* observations in a master training set. Then when each tree in the forest is grown, the training set for that tree is *n* observations chosen at random, with replacement, from the *n* observations in the master training set [27]. While maintaining the same distribution of categories as the master training set [27]. The second condition effects how many and which features are available for deciding how to split a node. When

determining a node split in a random forest, a random subset of the feature vector is selected for determining the best feature for the split [27]. This process produces a model that is more difficult to interpret than a decision tree, but provides a more generalized performance [28].

This section covered the various hardware and software components used to investigate a method of avoiding hazardous terrain with an autonomous ground vehicle. Effort was given to expose to the reader to key topics that are used in this research. The method of developing a random forest for terrain classification is discussed in Chapter III.

III. RANDOM FOREST

The random forest machine learning algorithm was chosen to classify terrain types in images due to its basis in the interpretable decision tree. Due to this basis it is possible to understand how each tree makes its classifications. Development of the random forest for this thesis work was conducted in several phases: data collection, feature extraction, growing the random forest, and evaluation. During data collection a set of raw images was gathered and sorted for use in growing the random forest. The feature extraction process required choosing a feature type and extraction scheme. Growing the forest requires using functions native to MATLAB, and deciding which optional arguments to include in developing the model. The evaluation process seeks to identify the predictive accuracy and relative speed of the classification process. The last three phases are iterative in that they are repeated until an acceptable solution is found—one that balances high predictive accuracy with relatively fast processing time.

A. DATA COLLECTION

To collect the training data, a script was written in MATLAB to automate the image-capturing process, saving each image under a distinct name. This script required the MATLAB Support Package for USB Webcams and is detailed in Appendix A. The robot was transported to the data-collection area on the Naval Postgraduate School campus, and the script was executed via secure shell access on a locally established network. As the robot was driven around the campus using the joystick controller, the script automatically captured images for use in our terrain classification algorithm. Effort was made to ensure each class of terrain in the area of operation was captured under the conditions the AGV would be operating. This meant driving the robot over the terrain at varying speeds, while traveling in a straight path, turning, and moving in and out of shaded areas.

The resulting set of images was sorted to create the training data. For this work, the training data is divided into two categories of terrain, hazardous and traversable, with each category having several subcategories. The hazardous terrain includes mulch, grass, and long grass. Traversable terrain includes sand path, concrete, and tiled concrete. The

categories were developed based on the work of Hargadine and Audette in [4] and [5], respectively. They each observed that the P3-AT experienced difficulties in maneuvering over hazardous terrain. This was due to ruggedness of the terrain and slippage of the wheels caused by travelling on them. The ruggedness caused the AGV bumpers to hit the ground and stop the system. The slippage degraded overall control of the system, and even resulted in the system getting stuck in terrain like mulch, where it can “dig” itself into the loose ground.

When sorting the training data, two types of images—nonhomogeneous terrain images and those that had portions of overexposure—were manually removed as they would have reduced the accuracy of the random forest classifier. Any overexposed portions had pixels with maximum intensity values, which provide no information about the terrain, rendering the features invalid. Mixed terrain-type images were not used for training the random forest but would be used later to test the efficacy of the random forest developed. Including images with these phenomena would have required identifying the portions with valid features, and ensuring they were labelled properly. This was not necessary as it was possible to develop a large number of observations with the homogeneous training data. Examples of each subcategory are shown in Figure 10. Examples of the types of images omitted from the training set are shown in Figure 11.

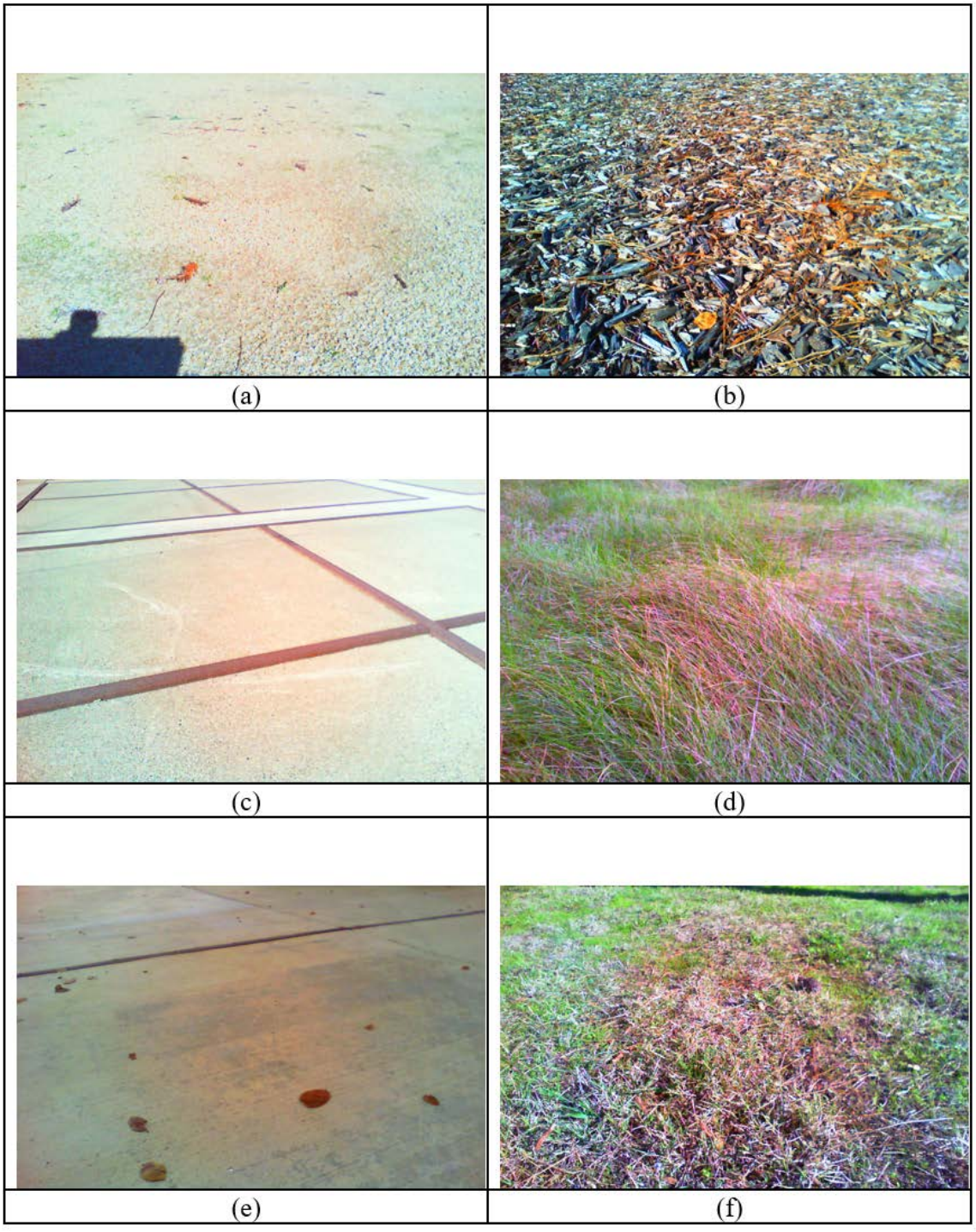


Figure 10. Examples of each subcategory of terrain: (a) sand path, (b) mulch, (c) tiled concrete, (d) long grass, (e) concrete, (f) grass

Figure 11. Examples of images omitted from the training set: (a) image with mixed categories and (b) image with overexposure

The resulting training data had 281 photos—146 of traversable terrain and 135 of hazardous terrain. The features used to train the random forest were extracted from these photos. The intent was to use the classification of terrain to develop a control effort for the AGV, which required labelling the observations as category "0" for traversable terrain and "1" for hazardous terrain. The post classification manipulation, discussed in Section IV.B, relies on using these categories.

B. FEATURE EXTRACTION

Features are descriptors that can be measured or calculated from a data set and can be used to train machine learning models. Based on the work described in [8] and the desire to use the classification results to avoid hazardous terrain, it was decided that features would be extracted from fixed points within each image. Inspired by a smaller sized grid from [8], it was chosen to separate the extraction points by 20 pixels both vertically and horizontally, with the outermost points located 10 pixels from the edge of the photo. The image dimensions were 800 pixels by 1280 pixels, which resulted in a 40-point by 64-point grid. This resulted in a total of 2,560 feature extraction points, which are shown as the blue crosshairs in Figure 12.

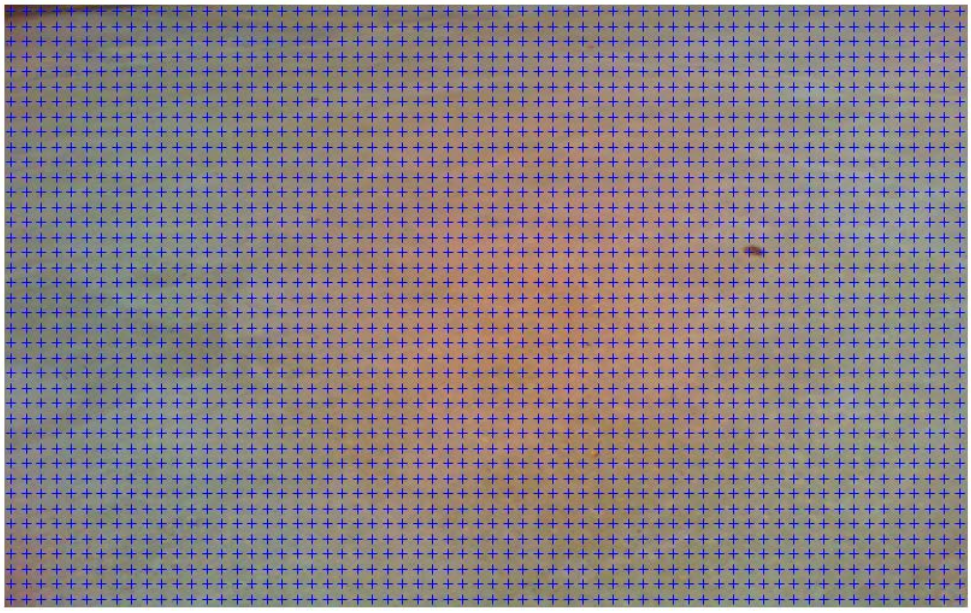


Figure 12. Feature extraction points displayed over a training image

Feature selection is affected by two primary performance concerns for the terrain classification algorithm: speed and accuracy. An accurate terrain classification ensured the desired control input was computed by the AGV, and relatively fast feature extraction influenced the overall classification speed. Classification speed is important as the objective was that the system behaves in a near-real-time manner, which is necessary when conducting motion planning and obstacle avoidance. These considerations affect which features are selected because a feature may provide very high predictive accuracy, but may take too long to extract for the system to behave in a near-real-time manner. The contrasting situation would be a feature that can be extracted very quickly, but provides predictions that cannot be used in controlling the AGV. Several feature types were explored including speeded up robust features (SURF) motivated by [31], pixel intensities, and pixel intensities with color representation. These feature types were used to grow random forests and determine their effect on predictive accuracy and the speed of the overall classification process. The results of laboratory and field tests, explained in Section III.D, led to the use of pixel intensities for feature extraction.

The pixel intensities were extracted from each image using the *extractFeatures* MATLAB function from the Computer Vision Toolbox. This function required an intensity image, a set of feature extraction points, and the extraction method. The intensity image comes from converting the image from the webcam to grayscale, and the set of extraction points are those shown in Figure 12. The "block" method was chosen as the extraction method. By using the "block" method, *extractFeatures* pulls the pixel intensities from the grayscale image at the extraction points. It was possible to set the size of the block extracted, but for this work the default size of 11 was used. The default value provided adequate results, regarding predictive accuracy and extraction speed, and was not explored further. This meant that for each fixed point, the function extracted the intensities of an 11 by 11-pixel region $\boldsymbol{A}$, and organized those values as a row vector. This vector is the feature vector $\boldsymbol{x}$, which contains the descriptors associated to the point in the image that $\boldsymbol{x}$ is extracted from.

A generic example of this process and results are pictographically represented in Figure 13, where the region $\boldsymbol{A}$ is a three by three region extracted from a five by five image centered on the element located at coordinate (3,3) that translates to the feature vector $\boldsymbol{x}$.

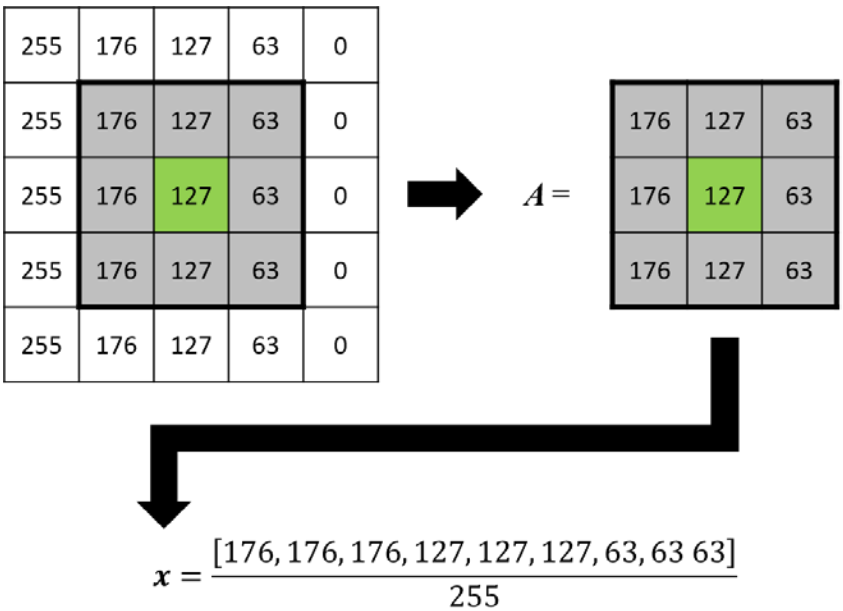


Figure 13. Graphic representation of the feature extraction process and results

As shown in Figure 13, the feature vector $\boldsymbol{x}$ was formed from the extracted region $\boldsymbol{A}$ centered on the coordinate of the fixed point. It was built by stacking the columns of $\boldsymbol{A}$, with the first column at the top and the last column at the bottom, and taking the transpose of the resulting column vector. The pixels were represented as unsigned 8-bit integers, and due to the data type required by the *predict* function, used later in this thesis, it was converted to double 64-bit format.

Given the size of the region used, the resulting feature vector $\boldsymbol{x}$ has 121 descriptors each having discretized values between 0 and 1. Then, element $\boldsymbol{x}_k$ of $\boldsymbol{x}$ is found according to

$$\boldsymbol{x}_k = \frac{\boldsymbol{A}_{i,j}}{255} \qquad 0 \le \boldsymbol{A}_{i,j} \le 255,\ \boldsymbol{A}_{i,j} \in \mathbb{Z}\,, \tag{1}$$

where k is

$$k = i + 11(j-1)\,, \tag{2}$$

and i and j are

$$1 \le i,\ j \le 11\,. \tag{3}$$

In Equation (1), $\boldsymbol{A}_{i,j}$ indicates the element of $\boldsymbol{A}$ from the i^{th} row and j^{th} column of the region.

While building the training set, the feature vectors were labelled after they were extracted according to the category it belonged to by appending a zero or one at the end of the feature vector. The labelled feature vector is the building block of the training set. By recursively loading images, extracting the features corresponding to the fixed points, and labelling them according to the category of the image, the training set was grown by 2,560 observations per image. This resulted in 719,360 observations for the collected data set and distribution of fixed points.

C. GROWING THE FOREST

The next step in developing a random forest for terrain classification was to grow the forest using the prepared training set. This provides a model that can be evaluated according to its predictive accuracy and speed. Growing the forest is accomplished by using the *TreeBagger* function from the MATLAB Statistics and Machine Learning

decision trees while using random portions of the features available to make splitting decisions [32]. Using *TreeBagger* with the training set generated a random forest for use in predicting terrain types in an image. The function required the number of trees, the training set, training labels, and the type of forest, classification or regression. There were several optional arguments available for building random forests with *TreeBagger*; this work used OOBPrediction and MinLeafSize. The minimum leaf size could be adjusted as a method of generalizing decision trees, and has the benefit of decreasing processing time of predictions [33]. The OOBPrediction argument allows analyzing the effectiveness of the random forest once it is grown without having to set aside a testing set [27], [33].

D. EVALUATION

Finding the acceptable forest composition began with growing 100 trees for various minimum leaf sizes and analyzing the classification error using the *oobError* function. This provided candidate compositions that could be analyzed further. These candidates were tested by implementing them on the AGV during scenarios designed to determine their accuracy and speed for short duration runs. This trial-based method yielded a forest that produced adequate classification results with sufficiently short processing time. The results of the various tests are detailed in Section V.A. The code for generating the random forest is included in Appendix B.

Once the forest was grown, it could be used to predict whether a set of features represented hazardous or traversable terrain. The prediction process required using the *predict* function in MATLAB. This function used the trained model and features extracted from the fixed points as inputs to classify each feature vector. Predictions made with random forests are based on the results of the classifications of the individual trees [27]. Each classification made by a tree is a vote for that category of terrain, and the majority vote is the forest classification [27]. Each tree makes a classification by comparing the appropriate feature values to the node splitting features and value until reaching a leaf, which corresponds to a classification [29]. The terrain prediction result is a vector with 2,560 elements, one for each extraction site, with a categorical “0” or “1” depending on the

terrain type predicted for the extraction site. Integrating the terrain prediction into the navigation solution is described in Chapter IV.

THIS PAGE INTENTIONALLY LEFT BLANK

IV. INTEGRATION OF VISION INTO THE NAVIGATION SOLUTION

Developing a method of avoiding hazardous terrain with vision was accomplished by addressing the entire navigation solution with all of its subordinate parts and not just obstacle avoidance. Navigating from point A to point B requires both obstacle avoidance and path planning. Obstacle avoidance requires sensing obstacles in the environment and reacting to avoid them en route to the goal. Path planning determines the route to the goal, and typically requires the robot to have some knowledge of the workspace it inhabits. The information the robot gathers from obstacle avoidance may be used to augment its knowledge of the workspace to improve path planning in future operations.

To be able to avoid hazardous terrain, the robot vision must be integrated into the navigation solution. This process begins with the treatment of the camera as a sensor. This chapter continues with how the terrain classification is filtered for use in the control mechanism. Once this is laid out, the process of controlling the AGV is examined under the context of a state-based machine. The remaining two sections address the use of simulation and obstacle oversight in this thesis work. Simulation was employed to investigate odd behavior of the AGV during testing and lead to developing and vetting certain methods used in the navigation solution. An additional result of investigating the odd behavior of the AGV was the need to provide oversight to the automatic memorization of certain obstacles encountered in the workspace, and this is discussed in the last section of this chapter.

A. CAMERA

To use the camera as a sensor for the control of the AGV it was necessary to mount it to see the areas of the reference frame of the robot that were most relevant to controlling its motion. Initially the camera was mounted to see to an expected distance of 2.0 meters from the AGV based on the success of [4] and [5] using the LIDAR to respond to convex obstacles at a range of 1.8 meters. This provided a field of view that captured relevant data, but it was then necessary to determine how to treat the camera input to build a control effort

from it. Inspired by the LIDAR, the camera input was treated as a point cloud, where each pixel was treated as coming from a distinct location in the reference frame. It was then necessary to properly correlate the imagery data to the space it imaged to produce useful controls for the AGV. Finding the portion of the reference frame projected into the images required approximating the field of view of the camera.

The field of view is approximated by starting with the assumption that the camera field of view is represented by the idealized geometry shown in Figure 14. This assumption primarily neglects imperfections in the setting of the imaging sensor and the asymmetry of the pixels about the axes of the camera. This abstraction also ignores the effects of the robot travelling over surfaces that induce pitch and roll, which would affect the portion of the reference frame that is projected into the image. Ignoring the imperfections of the camera and neglecting the induced pitch and roll allows the development of a simple model for using the photos from the camera to provide useful control input to the AGV.

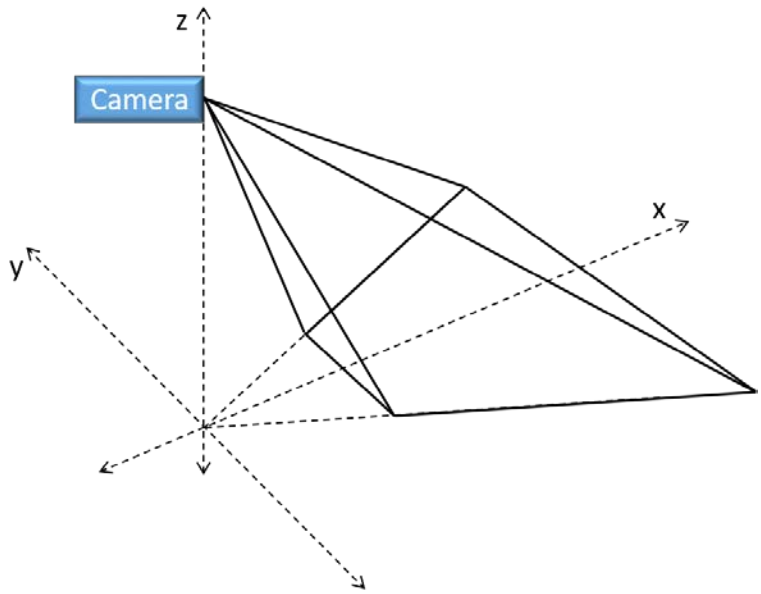


Figure 14. Idealized geometry of the field of view of the camera

To generate this model, photos were taken with the camera mounted on the AGV. These images were used to mark the horizontal and vertical limits of the field of view. The

vertical and horizontal limits were found over several iterations. Each iteration consisted of taking a picture and adjusting the position of the field of view limit markers until the limits were marked appropriately. Upon finding the limits, they were used to measure distances in the reference frame that were used to determine the field of view of the camera. This process is described throughout the remainder of this section. Finding the vertical field of view is based on the measured values and the planar geometry depicted in Figure 15.

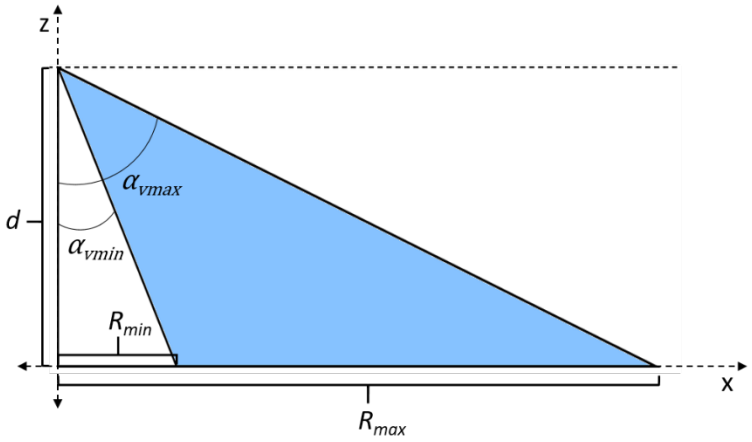


Figure 15. Geometry used to find the vertical field of view of the camera

The true vertical field of view is approximated using the distance R_{max} to the furthest viewed point, the distance R_{min} to the nearest viewed point, and d the height of the mounted camera. These allowed finding the angles α_{vmax}, and α_{vmin} using

$$\alpha_{vmax} = \arctan\left(\frac{R_{\max}}{d}\right), \tag{4}$$

$$\alpha_{vmin} = \arctan\left(\frac{R_{\min}}{d}\right), \tag{5}$$

and the vertical field of view is then

$$\alpha_{vfov} = \alpha_{vmax} - \alpha_{vmin}. \tag{6}$$

Determining the horizontal field of view α_{hfov} began with analyzing the idealized field of view represented in Figure 16.

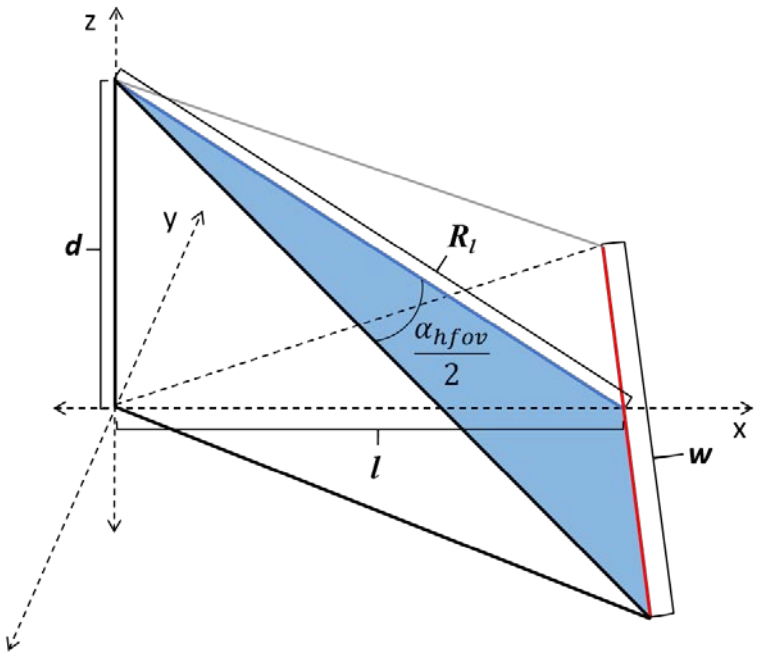


Figure 16. Geometry used to find the horizontal field of view of the camera

By using the trigonometric relationships between the measurable quantities *l*, *w*, and *d* it was possible to find α_{hfov} using

$$\alpha_{hfov} = 2\arctan\left(\frac{w/2}{R_l}\right), \tag{7}$$

where R_l is found according to

$$R_l = \sqrt{d^2 + l^2}\ . \tag{8}$$

Having determined the horizontal and vertical fields of view it was possible to determine the portion of the reference frame that was captured in each image. Finding the portion of the reference frame that corresponds to each pixel requires determining the orientation of each pixel and conducting a coordinate transformation from the image frame to the reference frame. This requires finding the angular height and width of the pixels, the mounting orientation of the camera, and the translation from the camera to the reference frame.

Determining the orientation of a pixel was simplified by assuming that the pixels have uniform angular height and width. The angular height of a pixel α_{ht} is found by dividing the vertical field of view α_{vfov} by the rows of pixels m in the image, such that

$$\alpha_{ht} = \frac{\alpha_{vfov}}{m}. \tag{9}$$

Similarly, it is possible to find the angular width α_{wd} using

$$\alpha_{wd} = \frac{\alpha_{hfov}}{n}, \tag{10}$$

where n is the columns of pixels in the image. This along with the mounting angle of the camera can be combined to find the orientation of each pixel.

Finding the mounting orientation of the camera started with the assumption that the orientation is constant with respect to time. This assumes the camera is mounted with a positive pitch down from the x-axis, while the roll and yaw are zero. Angles α_{vmax} and α_{vmin} can be used to find the measured mounting angle α_{mount} from

$$\alpha_{mount} = \alpha_{vmin} + \frac{\left(\alpha_{vmax} - \alpha_{vmin}\right)}{2}. \tag{11}$$

These angles are measured with respect to the line normal to the ground; however, the AGV uses an East-North-Up convention and the angles must be represented relative to the line parallel to the ground. Converting α_{mount} to θ_{mount}, the mounting pitch angle, depicted in Figure 17, is found by

$$\theta_{mount} = \frac{\pi}{2} - \alpha_{mount}\ . \tag{12}$$

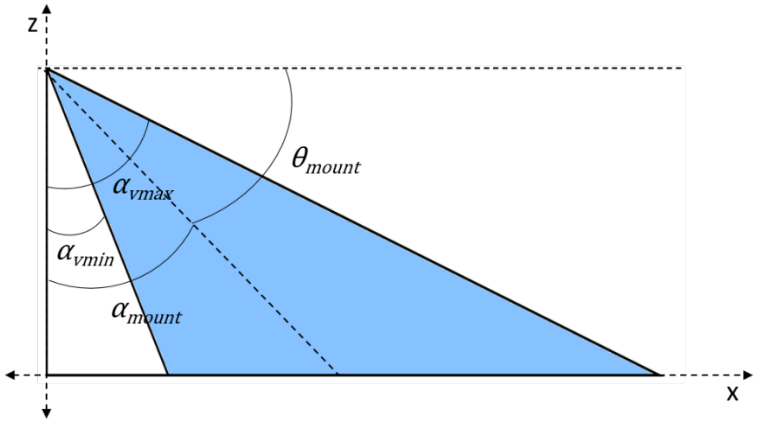


Figure 17. Diagram relating measured mounting angle to pitch of camera

Assuming images are symmetric about the camera line of sight enabled deriving the expression for the orientation of a pixel coming from the i[th] row and j[th] column in the image

$$\begin{bmatrix} \phi_{ij} \\ \theta_{ij} \\ \psi_{ij} \end{bmatrix} = \begin{bmatrix} 0 \\ \theta_{mount} - \alpha_{ht}\left(i - \frac{m}{2}\right) \\ \alpha_{wd}\left(\frac{n}{2} - j\right) \end{bmatrix}. \tag{13}$$

The range to the position captured in a pixel in the reference frame of the robot was found based on the assumption that the z-component is zero for all pixels. This assumption was made due to the intent of capturing images of the ground. The range was found using the geometry shown in Figure 18 where c_{ij} is the length of the hypotenuse of the triangle in the *xz*-plane, and was computed using the trigonometric relationship

$$c_{ij} = \frac{d}{\sin\left(\theta_{ij}\right)}, \tag{14}$$

and r_{ij} is the range to the pixel computed using the expression

$$r_{ij} = \frac{c_{ij}}{\cos\left(\psi_{ij}\right)}. \tag{15}$$

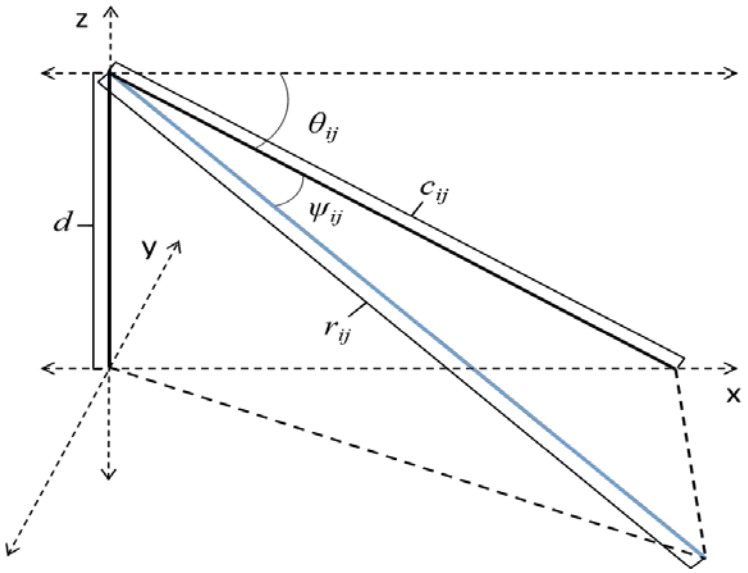


Figure 18. Diagram for calculating the range to a region captured in a pixel

Combining the range and orientation associated to a pixel allowed computing its position in the robot frame $[x_{ij}, y_{ij}, z_{ij}]^T$ by conducting the coordinate frame transformation described by

$$\begin{bmatrix} x_{ij} \\ y_{ij} \\ z_{ij} \end{bmatrix} = R_{ij}^{T} \begin{bmatrix} r_{ij} \\ 0 \\ 0 \end{bmatrix} - \begin{bmatrix} 0 \\ 0 \\ d \end{bmatrix}, \tag{16}$$

where R_{ij} is the rotation matrix corresponding to the orientation of the pixel

$$R_{ij} = \begin{bmatrix} \cos(\psi_{ij})\cos(\theta_{ij}) & -\sin(\psi_{ij}) & \cos(\psi_{ij})\sin(\theta_{ij}) \\ \sin(\psi_{ij})\cos(\theta_{ij}) & \cos(\psi_{ij}) & \sin(\psi_{ij})\sin(\theta_{ij}) \\ -\sin(\theta_{ij}) & 0 & \sin(\theta_{ij}) \end{bmatrix}. \tag{17}$$

The positions of the pixels corresponding to the robot frame were precomputed for use in determining repulsive forces from terrain and for finding the orientation of the terrain. The precomputed information was stored in two formats, as an array of *xy*-coordinates for each pixel and as range and bearing $[\rho_{i,j}, \beta_{i,j}]^T$ for each of the 2,560 points for which features are extracted from for terrain classification. The range and bearing to each feature extraction point is found using

$$\begin{bmatrix} \rho_{i,j} \\ \beta_{i,j} \end{bmatrix} = \begin{bmatrix} \sqrt{x_{ij}^2 + y_{ij}^2} \\ \arctan\left(\frac{y_{ij}}{x_{ij}}\right) \end{bmatrix}. \tag{18}$$

Figure 19 shows the calculated portion of the robot frame projected into the captured images.

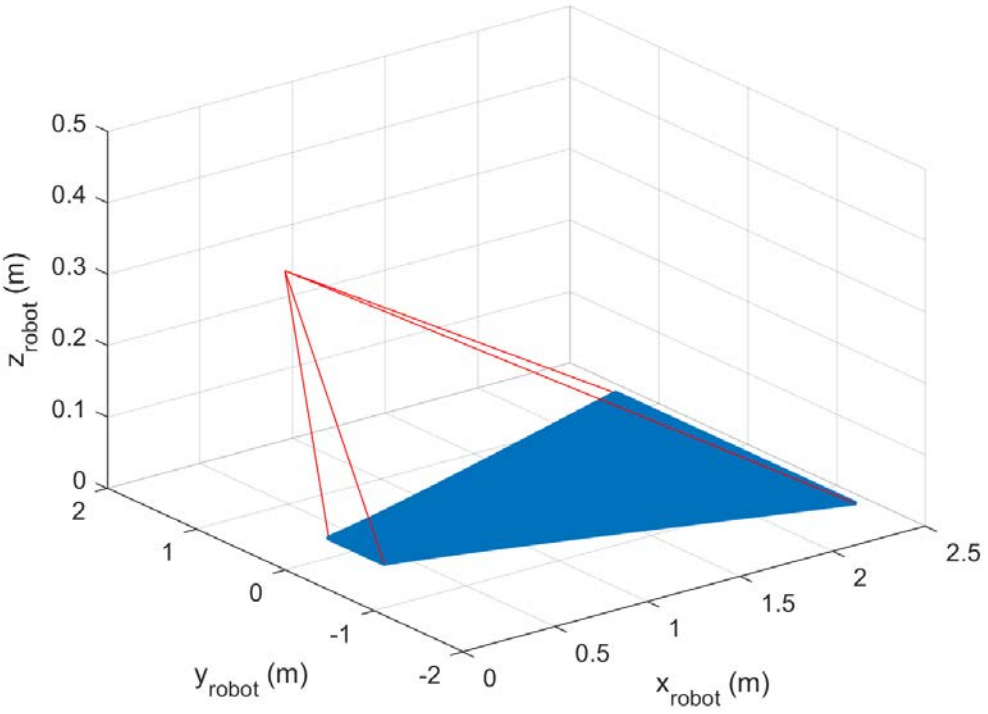


Figure 19. Projected portion of robot frame captured in camera images

Each fixed point now maps to a distinct point in the space that can be used to conduct obstacle avoidance. This treatment of the camera allows integration with the methods of terrain classification discussed in Chapter III, where a remaining concern is the inaccuracies, both false positives and false negatives, of the random forest, that may cause aberrant behavior if unmitigated. With this in mind the classification results are filtered to reduce the effects of false classifications.

B. FILTERING PROCESS

The prediction responses of the random forest, discussed at the end of Chapter III, are expected to have some degree of inaccuracy, and each falsely identified portion could generate control signals that turn into undesired behavior in the robot. Driven by an assumption that terrain is continuous in a way that means there will not be small patches of hazardous terrain in the middle of traversable terrain, and vice versa, the prediction results are filtered. False identifications are treated as noise to be filtered out. The prediction vector produced from the prediction process is reshaped into a 40 by 64 matrix, *PredMat*, such that each element aligns to the portion of the image the prediction comes from. *PredMat* is treated as being the true classification C_{true} plus noise ω, with values constrained to one or zero, such that

$$PredMat = \left(C_{true} + \omega\right) \bmod 2 . \tag{19}$$

Filtering *PredMat* is a two-step process involving a blurring and thresholding step followed by removing small regions of connected hazardous terrain. The blurring process is done by convolving *PredMat* with a three by three box filter γ; the resulting matrix is the average of each element in *PredMat* with the eight elements closest to it. The thresholding process rounds the elements to the nearest value, constraining values to zero and one. This process yields the intermediate matrix *PredTemp*:

$$PredTemp = \text{round}(PredMat * \gamma) . \tag{20}$$

The first step eliminates instances in which a few locations are identified as a terrain type opposite of the predominant type in that portion of the matrix, regardless of the classification. The second step involves removing small regions that have been classified as hazardous terrain. This was based on the fact that traversable terrain falsely classified as hazardous is considered more harmful to the motion planning effort than hazardous terrain classified as traversable terrain. This is done by eliminating regions classified as hazardous terrain that have 128 or less connected elements. In this case they are changed from Category 1 to Category 0. This was based on the assumption that terrain is continuous, and that regions this small would not occur outside of the corners of the image. This removal

of small regions assumed to be false positives is of greater importance to the control effort than reacting to terrain at the extremes. This process is accomplished using the *bwareaopen* MATLAB function from the Image Processing Toolbox. The processed classification matrix *ClassMat* is used to develop repulsive forces when hazardous terrain is identified. Examples of the results of the filtering process are shown in Section V.A. The code developed to classify images, process the classification matrix, and generate the controls from the terrain classification is included in Appendix F.

C. STATE-BASED FUNCTIONALITY

Up to this point, the focus of this chapter has been on the development of a model for using the camera data for sensing the environment and on filtering the classification predictions from the random forest. Now, focus is shifted to integrating that work into the previously existing control structure on the AGV. Previous thesis work, conducted by Hargadine in [4], developed the control structure for the AGV as a state-based machine. Each state has defined goals and control mechanisms to achieve those goals. The state diagram that defines the behaviors of the AGV is shown in Figure 20. This diagram is an evolution of the work done in [4] and [5], and accounts for the capabilities added by this work.

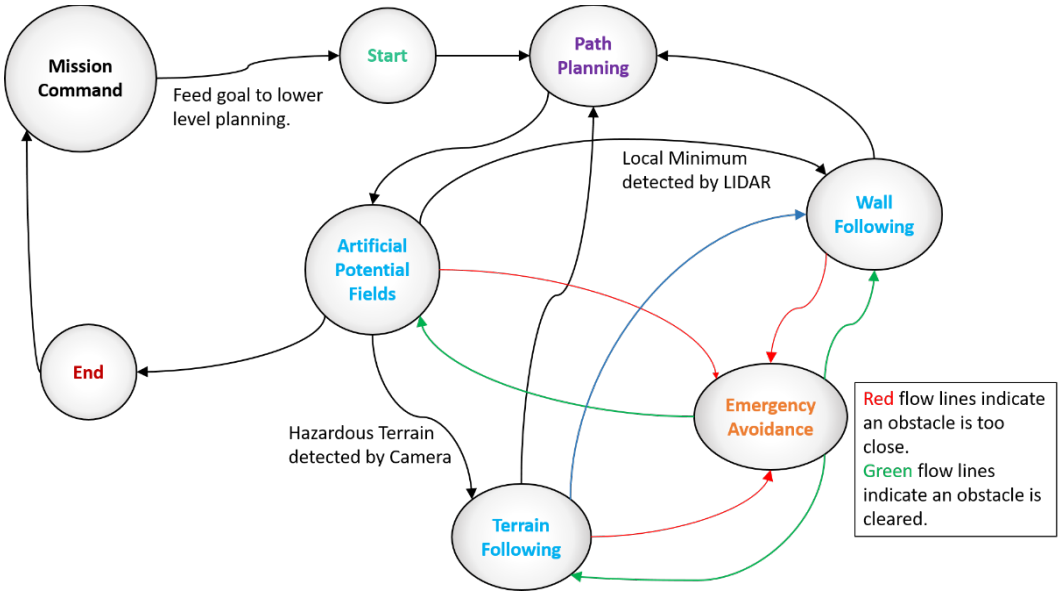


Figure 20. State diagram defining AGV operations

The Mission Command state is based on the work of Audette in [5], and feeds the location of the current goal to the Path Planning state. The Path Planning state uses the information known about the workspace to plan an optimal route to the goal, which is executed by the Artificial Potential Fields state. The Artificial Potential Fields state employs a gradient descent mechanism to maneuver toward the waypoints along the path and ultimately toward the goal. Based on its interactions with the environment, the system can transition into several additional states: Terrain Following, Wall Following, and Emergency Avoidance. Each of these states is addressed further in this chapter, beginning with the Path Planning state. The code that executes the state-based functionality is represented in Appendices C and D.

1. Path Planning

Path planning is employed by the robot to get from the initial position to the goal position while avoiding known obstacles in the workspace. This requires a representation of the workspace with the locations of obstacles and a method for planning the path. This thesis work makes use of the approximate cell decomposition method to represent the workspace. With this representation, the A* algorithm is used to find the optimal route to the goal. This process begins with the implementation of the approximate cell

decomposition on two scales, to refine known obstacle position and for path planning, and then the method of implementing the A* algorithm is discussed.

a. Approximate Cell Decomposition

The approximate cell decomposition process serves two functions in this thesis work. The first is that it takes the location of known obstacles and refines them, and the second is to provide a representation of the workspace that can be used to plan a path. The workspace representation generated by approximate cell decomposition is the map the robot uses to determine its route to the goal. This map only contains the location of obstacles it has interacted with in previous operations. As described in Section II.C.1, the standard application of the approximate cell decomposition methodology requires subdividing cells that are partially occluded by obstacles. This is not done in this work as obstacles are identified as points in the workspace rather than two-dimensional objects, which would require dividing cells down to infinitely small sub-cells to contain a single point obstacle. Also, based on the AGV size and knowledge of the workspace there is a practical limit to how small the cells should be. By initially using a larger cell size, it is possible to refine the perceived position of known obstacles. While using a smaller scale, the decomposition of the workspace can be used for path planning.

Approximate cell decomposition identifies the cells that are occupied by obstacles and the cells that are perceived to be free to move through. To build this representation the robot uses its knowledge of the workspace and the obstacles it has encountered in the past. It was noticed that the location of previously identified obstacles, terrain or other local minimums, were not precisely located. This was primarily due to the inaccuracy of the GNSS/INS localization. The location of each known obstacle is modelled as the true location plus noise

$$\begin{bmatrix} x_{known} \\ y_{known} \end{bmatrix} = \begin{bmatrix} x_{true} \\ y_{true} \end{bmatrix} + \begin{bmatrix} \nu_x \\ \nu_y \end{bmatrix}, \tag{21}$$

where the noise, ν_x and ν_y, are assumed to be normally distributed with zero mean, and have an unknown standard deviation σ that is

$$\nu_x, \nu_y \sim N(0, \sigma). \tag{22}$$

To estimate the true position of known obstacles it was assumed that terrain tends to remain continuous, as described, and that the boundaries between terrain types are linear. This allows looking at the identified locations of the known obstacles as points that describe an unknown line, the line that describes the boundary between traversable and hazardous terrain.

This boundary line is found by using an approximate cell decomposition of the known workspace at a larger scale, dividing it into four-meter by four-meter cells. This representation is used to fit a line to any set of obstacles in a cell containing three or more known obstacles. Figure 21 represents an example of a workspace with point obstacles that has been decomposed as described. These cells are inspected for obstacles, and if there are more than two obstacles in a cell, a least squares fit is conducted to approximate the line that those obstacles form. This results in two primary effects. First, the identified boundary is less noisy, providing for more accurate path planning and more even control when included in the artificial potential fields. Second, some shapes, particularly corners, are misconstrued. This process provides a needed refinement to the identified position of obstacles that feed the workspace representation used for path planning.

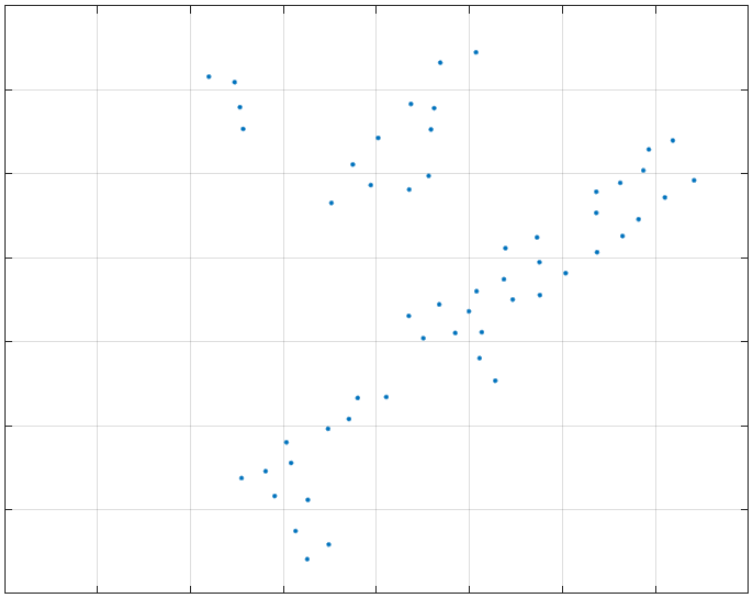


Figure 21. Example approximate cell decomposition with four-meter-square cells and point obstacles

When planning a path from point A to point B, with knowledge of obstacles between the two points, it is implied that the path should avoid the obstacles, and approximate cell decomposition is a starting point in planning this path. The scale at which this is done for refining the location of obstacles is not reliable for the path planning task due to the size of the cells compared to the paths available in the workspace. As a result of this, the workspace is readdressed using the refined positions of the known obstacles.

For the path planning application, the workspace is divided into one meter by one-meter cells, called the small-scale representation (SSR). The cells are small enough to plan paths around the obstacles yet large enough the AGV can move through them without issues. The SSR will be used to find a path from the initial position to the goal position, and it was beneficial to assume there are unknown portions of the workspace. The known workspace Ω consists of the initial position q_{init}, goal position q_{goal}, and the refined position of the known obstacles. This is padded to account for the unknown portions of the workspace, and defined as Ω_{padded}. Padding allows the AGV to avoid trivial geometries that would prevent finding a path from q_{init} to q_{goal}. A trivial geometry is shown in Figure 22, where if a path planner only knew of the shaded portion of the workspace it would be impossible to find a path to the goal. Whereas, if this were expanded a small amount, then an obvious path around the obstacle becomes available to the planner.

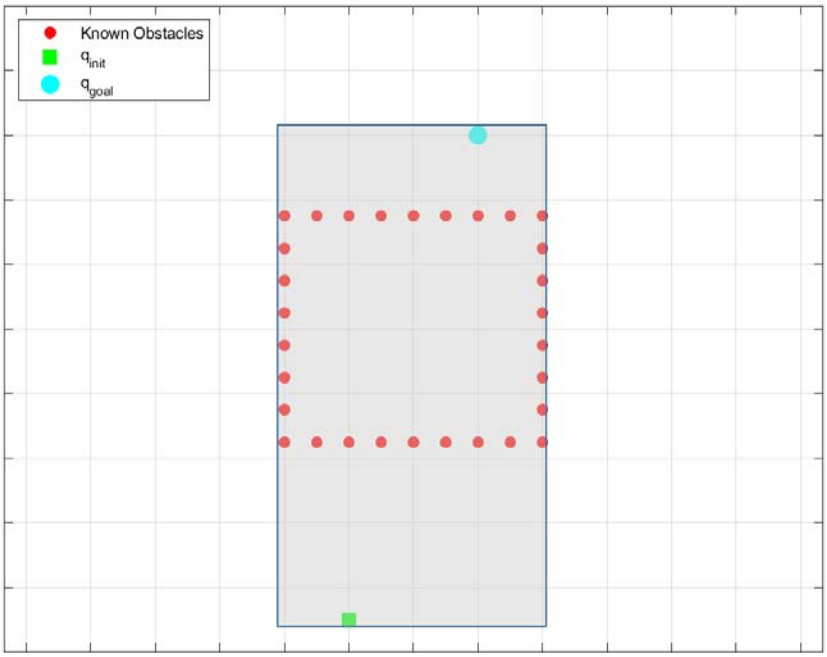


Figure 22. Trivial workspace geometry requiring padding to find a path from q_{init} to q_{goal}

The padded SSR Ω_{padded} is used to plan the path to the goal by using the A* search algorithm. Code developed for generating the workspace representation and refining obstacle locations is included in Appendix E.

b. A Implementation*

To implement the A* search algorithm for path planning a MATLAB script was written that accounts for the occupied cells, the heuristic cost, the cost associated with a cell, those cells that have been visited, and the path to a cell. These are tracked using matrices *O, H, F, V,* and cell array *Path,* respectively. Each matrix is established with zeros for all elements, the cell array is established with empty cells, and each matrix/cell array has the same dimensions as the padded SSR generated for path planning. The elements of the matrices and cell array correspond to cells in Ω_{padded}; that is information about the cell located in the i^{th} row and j^{th} column of Ω_{padded} is found in the i^{th} row and j^{th} column of the matrix/cell array.

The occupancy matrix O is developed from Ω_{padded} and assigns a value of one to those elements of O that correspond to cells occupied with obstacles in Ω_{padded}, and zero otherwise. This matrix is treated as a black and white image that is morphologically manipulated to close small gaps and ensure a path is not planned through diagonally connected free space. The modified occupancy matrix $O_{modified}$ now indicates there are obstacles where there were previously small gaps and diagonal connections. This process is detailed in the *AlphaStar.m* script included in the Appendix E. This matrix is used to build the H matrix.

When building H, any corresponding elements of $O_{modified}$ that indicate an open cell are assigned a heuristic cost equal to the Euclidean distance from the midpoint of the cell to the goal. Any cells in the modified occupancy matrix with obstacles are assigned a cost of infinity in the corresponding location in H. The exception to this is if the initial or goal position happens to reside in a cell that contains a known obstacle, the known obstacle is ignored, and the cost is equal to the Euclidean distance. This presents the possibility for paths to be planned through known obstacles, and in this case the online controls are trusted to avoid them. Equation (23) describes the general cost assignment to the H matrix elements, where $p_{mid(i,j)}$ is the midpoint of the cell in the ith row and jth column of Ω_{padded},

$$H_{i,j} = \begin{cases} \sqrt{\langle p_{mid(i,j)}, q_{goal} \rangle} & if \ O_{i,j} = 0 \\ \infty & if \ O_{i,j} = 1 \end{cases}. \tag{23}$$

By starting in the initial cell, containing q_{init}, the algorithm searches for the optimal path to q_{goal}. The search process is recursive and requires checking the neighbors of the cell being "visited." Checking a cell in this case generates the corresponding element of *Path* and F.

The element of *Path*, for the checked cell, is built by using the element of *Path* of the "visited" cell and adding the midpoint of the cell being checked to the end. The element of *Path* corresponding to the initial cell is q_{init}. Each element of *Path* becomes a list of coordinates that can be used as waypoints to get from q_{init} to the cell.

The cost associated with a cell is $F_{i,j} = g(i,j) + H_{i,j}$, similar to the method described in Section II.C.2. The function *g(i,j)* is the cost to get to cell *i,j* from the initial cell through the path defined in the element of *Path*. If the cell is being checked for the first time or if the cost along the current path is cheaper than the previous checks the cost and path are updated, and otherwise the cost and path from checking is ignored. Once all neighbor cells have been checked, the matrix *V* is updated to indicate the current cell being "visited" has been "visited." The next step is to determine which cell to "visit" next. This is done by checking the matrix *F* for the cheapest cost of an unvisited cell. The process begins anew upon identifying the next cell to "visit."

Upon finding a path to q_{goal} the *AlphaStar.m* script post processes the path. This is based on the fact that the path is likely to have more points than are necessary to effectively navigate to the goal. This required downsampling the identified path so that every fourth coordinate pair in the path list is kept for use in navigation. This downsampled path is passed to the AGV for use as waypoints to the goal.

With a path planned from the current position to the goal, the route is executed using the artificial potential fields algorithm, and it is at this point that the AGV transitions from the Path Planning state to the Artificial Potential Fields state. The Artificial Potential Fields state is described in the next section. The full code for implementing the path planning is provided in Appendix E.

2. Artificial Potential Fields

As was alluded to earlier, artificial potential fields is a well-known navigation methodology. According to Jean-Claude Latombe in [9], this method assumes the robot is a point mass that is effected by a potential field *U* that pulls the robot to the position of least potential energy. The overall potential field is treated as the sum of attractive U_{att} and repulsive U_{rep} potentials. Repulsive potentials provide obstacle avoidance capabilities by assigning areas near obstacles a higher potential than the surrounding area. Taking the negative gradient of the potential field function at point *q*

$$F(q) = -\nabla U(q) \tag{24}$$

generates the force F at q that avoids local obstacles while maneuvering closer to the goal [9]. The artificial potential field algorithm has inherent limitations that must be considered. The most relevant to this work is that of local minima. The method of implementation is covered throughout this section beginning with the attractive potential and resulting force.

a. Attractive Force

The method of implementing the attractive potential and resulting force matches that documented in the previous thesis work done by Hargadine in [4], and is briefly covered for completeness. The method is inspired by Latombe's description of a parabolic potential in [9], where the potential field U_{att} at point q is

$$U_{att}(q) = \frac{1}{2}\xi\rho_{goal}^2(q), \tag{25}$$

where ξ is a positive gain that can be tuned to adjust the performance of the AGV, and ρ_{goal} is the Euclidean distance to the goal from q.

A characteristic of the parabolic well is that the forces at extreme distances from the goal extends toward infinity which is undesirable. To avoid this the resulting attractive force has a bounded magnitude to ensure the AGV operates at a safe speed. A safe speed is defined by responding well to repulsive forces, and does not endanger the platform. The repulsive force is expanded from the work in [4], and is described in the next section.

b. Repulsive Force

The repulsive force implemented in this thesis builds on the work done by Hargadine by adding methods of sensing obstacles. The previous work uses the description of a repulsive potential described in [9] as

$$U_{rep}(q) = \begin{cases} -\frac{1}{2}\eta\left(\frac{1}{\rho(q)} - \frac{1}{\rho_0}\right)^2 & \text{if } \rho(q) \le \rho_0 \\ 0 & \text{if } \rho(q) > \rho_0 \end{cases}, \tag{26}$$

which has a range of influence defined by ρ_0 and a gain η for tuning behavior near obstacles, and $\rho(q)$ represents the range to the sensed obstacle. Equation (26) is the basis for generating the repulsive forces used to avoid obstacles.

The repulsive force is the result of the AGV perception of obstacles by the LIDAR and camera as well as knowledge of nearby known obstacles. Each component is developed similarly, but with consideration for the ways obstacles are represented by the sensing mechanism.

The LIDAR component is built upon the idea of a point cloud where each return contributes to the total LIDAR force using the standard repulsive force model as seen in [9]. The total LIDAR force F_{LIDAR} implemented in [4], and used in this work, is

$$F_{LIDAR} = \sum_{i=1}^{1032} -\eta_{LiDAR} \left(\frac{1}{\rho_i} - \frac{1}{\rho_{L0}} \right) \left(\frac{1}{\rho_i^2} \right) \begin{bmatrix} \cos(\theta_i) \\ \sin(\theta_i) \end{bmatrix}, \tag{27}$$

where η_{LIDAR} is a scalar gain for the LIDAR generated force, ρ_i is the distance associated with the ith return, ρ_{L0} is the maximum detection distance that contributes to the repulsive force, and θ_i is the angle associated with the return.

The force from known obstacles F_{Known} is a new addition to the AGV capabilities. This was initially implemented to avoid areas where terrain had been identified and uses the standard repulsive force potential. Due to observed behavior in operation the force was simulated in MATLAB; the simulations are discussed in Section IV.D. Those simulations revealed that constraining the input of the force to the three known obstacles closest to the AGV provided smoother behavior. This was carried over to the implementation on the AGV, and the force component from known obstacles is generated according to

$$F_{Known} = \sum_{i=1}^{3} -\eta_{Known} \left(\frac{1}{\rho_i} - \frac{1}{\rho_{K0}} \right) \left(\frac{1}{\rho_i^2} \right) \begin{bmatrix} x_i - x_R \\ y_i - y_R \end{bmatrix}. \tag{28}$$

Equation (28) is similar to Equation (27) used for the LIDAR but has its own range of influence ρ_{K0} and gain η_{Known}. This differs slightly from the method used for the LIDAR in that it generates the force based on the relative position of the obstacle $[x_i, y_i]^T$ from the AGV position $[x_R, y_R]^T$.

The force generated from the terrain classification is conceptually similar to the LIDAR force in that the image is treated as a point cloud. *ClassMat*, generated as described in Section IV.B, is used to determine which points in the image have hazardous terrain,

and those points are used to build $F_{Terrain}$. In this case, each point was considered according to its precomputed range and bearing from the AGV position $[\rho_{i,j}, \beta_{i,j}]^{\mathrm{T}}$ as found in Section IV.A. Due to the limited range, greater number of returns, and the space those returns occupy in the reference frame of the robot, the force contribution is altered. The terrain force is found using

$$F_{Terrain} = \sum_{i=1}^{40}\sum_{j=1}^{64} -\eta_{Terrain}\left(\frac{1}{\rho_{i,j}} - \frac{1}{\rho_{T0}}\right)\begin{bmatrix} G_x \cos(\beta_{i,j}) \\ G_y \sin(\beta_{i,j}) \end{bmatrix} \tag{29}$$

with a gain for overall tuning $\eta_{Terrain}$ and range of influence ρ_{T0} similar to the previous repulsive forces mentioned but includes individual component gains G_x and G_y for tuning behavior based on the field of view. It also drops the proportionality of the force to the inverse square of the distance to the observation. The removal of that proportionality provided more even distribution of the force over the depth of the field of view. The code for generating the repulsive forces is included in Appendix F.

The total repulsive force is the sum of the three forces developed above, that is

$$F_{Repel} = F_{LiDAR} + F_{Known} + F_{Terrain}\,. \tag{30}$$

Throughout the discussion above, the term force is used, and this is primarily a matter of custom. In implementation, the total "force" is used as velocity commands for a robot. With proper tuning, this method typically allows the AGV to move from its initial position to the goal with little to no information about the workspace; however, there are limitations to the artificial potential fields method that can prevent the AGV from reaching its goal. Of particular interest to this work is the issue of local minima. The methods used to handle this phenomenon are discussed in the next section.

c. Escaping Local Minima

Local minima refer to any location in the artificial potential field surrounded by higher potentials other than the goal. These locations can trap the AGV. This topic is addressed in [34] and [35] using two different approaches. In [34], the local minima are dealt with by invoking a global path planner, while in [35], the obstacle responsible for the

local minimum is followed until the minimum is escaped. In this thesis these methods are modified and combined to escape local minima and not return to them.

The process of handling local minima starts with detecting it. This thesis assumes two types of obstacles generate local minima, convex obstacles and terrain. Each is identified, confirmed, and handled in different ways. Upon confirming a local minimum, the location of that minimum is added to an array of known obstacles in the AGV databanks, which are used for future path planning and as part of the known obstacle force as described in Sections IV.C.1 and IV.C.2.b. Then the AGV executes an escape mode depending on the type of local minimum identified. If the minimum is due to convex obstacles, then the system uses a wall-following algorithm to escape. This method is developed and discussed further in [4], and is briefly discussed here for completeness.

(1) Wall Following

The Wall Following state was entered when the AGV became trapped in a local minimum due to convex obstacles. These local minima were identified and confirmed when the force due to the attractive force and LIDAR force was less than a threshold τ_{LIDAR}. The escape algorithm followed the boundary of the obstacle forming the local minima by keeping it perpendicular to the AGV. This may cause the AGV to get farther from the goal in the process. The system determined the direction to follow the obstacle by referencing the direction it was pushed by the LIDAR force prior to entering the Wall Following state. To escape the Wall Following state, the system tracked its distance to the goal g_{dist}, and compared this to g_0, which started as the distance to the goal upon entering the Wall Following state. Each time $g_{dist} < g_0$ then g_0 was set equal to g_{dist} and a count variable d_{count} was incremented. When d_{count} exceeded a tunable threshold, the system exited the Wall Following state. Local minima due to terrain were dealt with differently than convex obstacles.

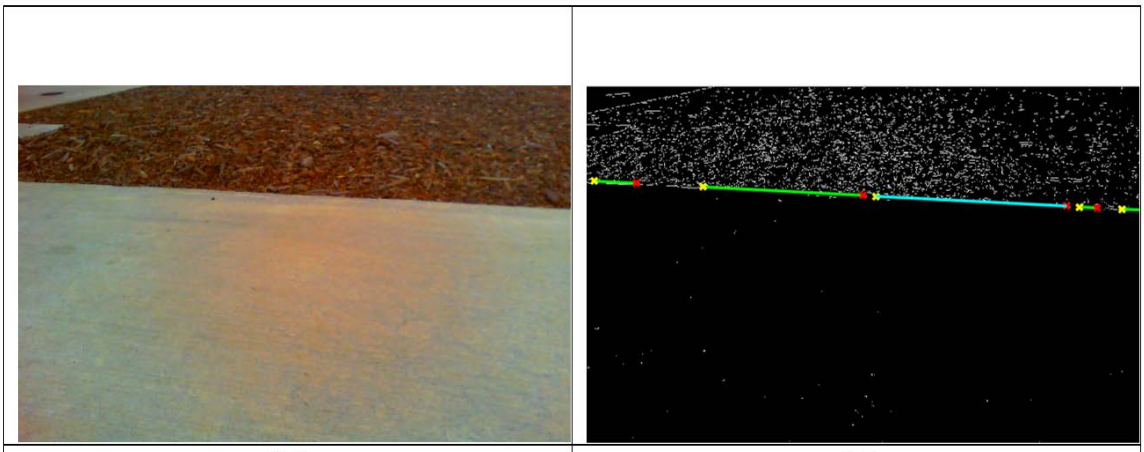
(2) Terrain Following

Potential local minima due to terrain were identified whenever the terrain force exceeded a threshold $\tau_{Terrain}$. Confirming a minimum required that the previous iteration

identified a potential minimum, and the terrain force generated on that iteration was in excess of $\tau_{Terrain}$. Upon confirming the local minimum, the AGV entered the Terrain Following state. The narrow field of view of the camera and the reliability of classification made it infeasible to keep the terrain perpendicular to the AGV as was done in the Wall Following state. One analogous approach was to determine the orientation of the terrain and place a temporary goal at a set distance from the AGV along that orientation, essentially following the orientation of the terrain for a short distance. This required determining the orientation of the terrain, which was accomplished using the Hough transform. It also required determining which side of the AGV the temporary goal would be placed along this orientation.

(3) Finding the Orientation of the Terrain

Finding the orientation of the terrain was accomplished by using the Hough transform in a multi-step process that began with capturing an image immediately after entering the Terrain Following state. The next step was an edge detection algorithm that yields the edges in the image. For this thesis, the Sobel method was used. The Sobel method entails passing the image through a filter that approximates a horizontal gradient as well as a filter that approximates a vertical gradient. The resulting gradient images are then summed and subjected to a threshold, such that values less than the threshold are set to zero. This edge image is analyzed in the ρ-θ space to determine if lines exist in the image, and if so where the most discernable lines are. This process is accomplished in MATLAB by using the series of functions from the Image Processing Toolbox: *edge*, *hough*, *houghpeaks*, and *houghlines*. This generated the standard Hough transform for the image and the five most prominent lines in the image were used to determine the orientation of the terrain in the image ψ_{image}. Figure 23 shows an example of an image and the results of the standard Hough transform.



(a) (b)

Figure 23. Example image and the resulting Hough Transform information overlaid on the results of edge detection process

Upon having ψ_{image} and the precomputed locations associated with each pixel, found in Section IV.A, it was possible to determine the orientation of the terrain $\psi_{Terrain}$ relative to the robot.

(4) Placing the Temporary Goal

To escape the local minimum induced by terrain, it was chosen to place a temporary goal 2.5 meters from the robot in the direction of the orientation of the terrain. This distance allowed the robot to avoid the terrain it had encountered, and it was also not so far that it allowed the overall mission to be neglected. This did yield two possibilities for placing the temporary goal along the orientation, left or right of the robot. This choice was made according to the direction it had been pushed by the terrain force. The robot then places the temporary goal according to

$$\begin{bmatrix} x_{temp\_goal} \\ y_{temp\_goal} \end{bmatrix} = \begin{bmatrix} 2.5\cos\left(\psi_{Terrain}\right) \\ 2.5\sin\left(\psi_{Terrain}\right) \end{bmatrix}, \tag{31}$$

where $\psi_{Terrain}$ accounts for any adjustment required for turning left or right of the AGV.

The AGV then uses artificial potential fields to travel to the temporary goal. Upon reaching the temporary goal, the system exits the Terrain Following state. After exiting

either local-minimum-escape modes the system clears any preexisting path and enters the Path Planning state before returning to the Artificial Potential Fields state with a new route to the goal.

The system is equipped with an Emergency Response state that reacts to being too close to obstacles, and was implemented for safety purposes. The Artificial Potential Fields state is not well suited for handling dynamic obstacles, such as pedestrians, and in the event something or someone were to abruptly approach the robot, the Emergency Response mode was able to avoid collisions. Regardless of the state the AGV is in, if three or more LIDAR returns indicate an obstacle is within half a meter it will enter emergency mode. This mode causes the AGV to stop for four seconds and then reverse for four seconds. It then rechecks the area and, if necessary, executes the same protocol. Otherwise, it returns to the state it was in prior to entering the Emergency Response state.

This completes the description of how the vision was integrated into the AGV architecture. This leaves two areas important to this work undeveloped. The first is the simulation process, which led to refining the known obstacle positions, determining the number of obstacles to include in the known obstacle force, and developing the path planning process. The second is the fact that the system required operator oversight as it developed awareness of the workspace. As the system identified locations of obstacles, it was known that there was noise in the positions, and this was typically mitigated through refining those positions. In some cases, refinement was insufficient to mitigate the degree of error in the obstacles position, and in those cases the obstacles had to be removed from the AGV databanks. These topics are addressed in the next sections

D. SIMULATION

Simulating the effects of the known obstacles led to the development of many of the techniques used throughout the navigation solution proposed by this thesis, such as refining obstacle location and developing the path planning algorithm. The simulation was built in MATLAB as an implementation of the artificial potential field algorithm with the simulated robot reacting solely to known obstacles. Due to the inherent limitations of the artificial potential field algorithm discussed in [34], the simulation was outfitted with a

method analogous to the wall-following mode discussed in [35] to escape local minima and mimic the AGV capabilities.

The simulations emulated the robot maneuvering from the opposite side of an impassable region toward its goal location. For these simulations the obstacle region was described by points on the edge of the region. This set of obstacles was used in the simulation to examine the behavior of the robot under several scenarios, and is visualized in Figure 24 along with the simulated initial position of the robot and goal location. This process enabled resolving initial parameters for the known obstacle force for real world testing, and led to discovering the benefits of obstacle refinement and path planning.

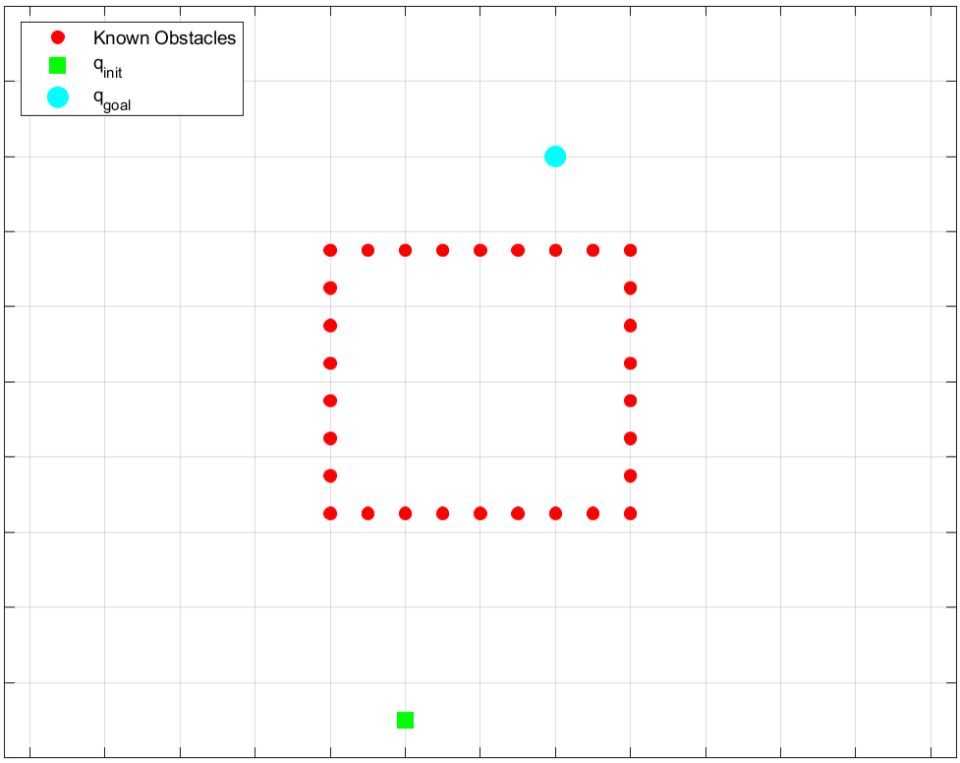


Figure 24. Representation of true obstacle position for use in simulation

The initial simulations used the known obstacles at the true locations, and they were followed by simulations where the obstacles had noisy positions. The behavior of the simulated robot with noisy obstacle locations was considered similar to that experienced with the AGV prior to conducting the simulations.

The next simulation implemented the large-scale approximate cell decomposition with least squares fit to refine the obstacle locations. In addition to examining the effects of refining the noisy obstacle location, this simulation was used to investigate the known obstacle force contribution, the range of influence, and the repulsive gain for known obstacles. The simulations were analyzed with a focus on the quality of the trajectory and obstacle refinement as nearly all resulted in the simulated robot reaching the goal position. Quantitative analysis was restricted to how close the simulated robot got to the known obstacles.

The final set of simulations were run using noisy obstacles, refined as described, and included the path planning algorithm using A* search. The results of the simulations are discussed in Section V.C. This provided insight into the ability to refine the known obstacle positions, but hinged on the fact that obstacles were reasonably noisy. Those obstacles identified in locations that were beyond the reasonable threshold were dealt with by operator oversight; this process is discussed in the next section.

E. OBSTACLE OVERSIGHT

While it was possible to handle the noisy placement of obstacles in liminal spaces there were times the system memorized locations of obstacles that were erroneous beyond noise. This led to manually inspecting the obstacles the AGV had in its memory. The erroneous locations were typically due to the localization being vastly off base, but was also observed due to egregious misclassification of terrain.

To check the locations of the known obstacles, they were converted to latitude and longitude values using the AlvinXY projection. This method is discussed in [36], and requires using a reference point to serve as the local origin to determine the approximate linear distance of a degree in latitude m_{deglat} and longitude m_{deglon} by using

$$m_{\text{deglat}} \approx 111132.09 - 566.05\cos(2\phi_0) + 1.2\cos(4\phi_0) - 0.002\cos(6\phi_0) \qquad (32)$$

and

$$m_{\text{deglon}} \approx 111415.13\cos(\phi_0) - 94.55\cos(3\phi_0) - 0.12\cos(5\phi_0), \qquad (33)$$

where ϕ_0 is the latitude of the reference point. This was used to convert the position of obstacles (x_i, y_i), stored as meters from the origin in the rectilinear plane to latitude and longitude (ϕ_i, λ_i) with the reference point as the origin using

$$\phi_i = \frac{y_i}{m_{\text{deglat}}} + \phi_0 \tag{34}$$

and

$$\lambda_i = \frac{x_i}{m_{\text{deglon}}} + \lambda_0 , \tag{35}$$

where λ_0 is the reference longitude [36].

Upon converting the known obstacle locations to latitude and longitude, it was then possible to store them as a KML file using the *kmlwritepoint* function, part of the MATLAB Mapping Toolbox. The file generated was then viewed in Google Earth Pro to verify the position of known obstacles. Removing erroneous obstacles only required deleting the offending coordinates from the AGV databanks.

THIS PAGE INTENTIONALLY LEFT BLANK

V. EXPERIMENTS AND RESULTS

The methods described in Chapters III and IV provided a means of identifying and avoiding hazardous terrain. The results of this design solution are presented and analyzed throughout this chapter. This chapter starts with the growth and analysis of several random forests designed for terrain classification. This is followed by the results of the filtering process used to reduce misclassifications. From there the results of the simulations described in Section IV.D and their ramifications are discussed. Finally, real-world trials of the fully integrated navigation solution are presented and discussed to conclude this chapter.

A. TERRAIN CLASSIFICATION

The first goal of the thesis research was developing a machine learning algorithm to classify terrain, and the random forest machine learning algorithm was selected to accomplish this. Determining a random forest that provided accurate results with fast prediction speed started with growing several forests and examining the error associated with each. This formed the basis of deciding the composition of forests to be tested for acceptable processing times when implemented on the AGV. For this research, 12 forests were grown, each with 100 trees. The trees in the test set had minimum leaf sizes (MLS) of 1, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50, and 55. The error associated to each forest was examined by using the *oobError* function from the MATLAB Statistics and Machine Learning Toolbox. The *oobError* function provides an unbiased estimation of the classification error by using out-of-bag observations [33]. The results of several of the forests are shown in Figure 25, and the information is representative of the set.

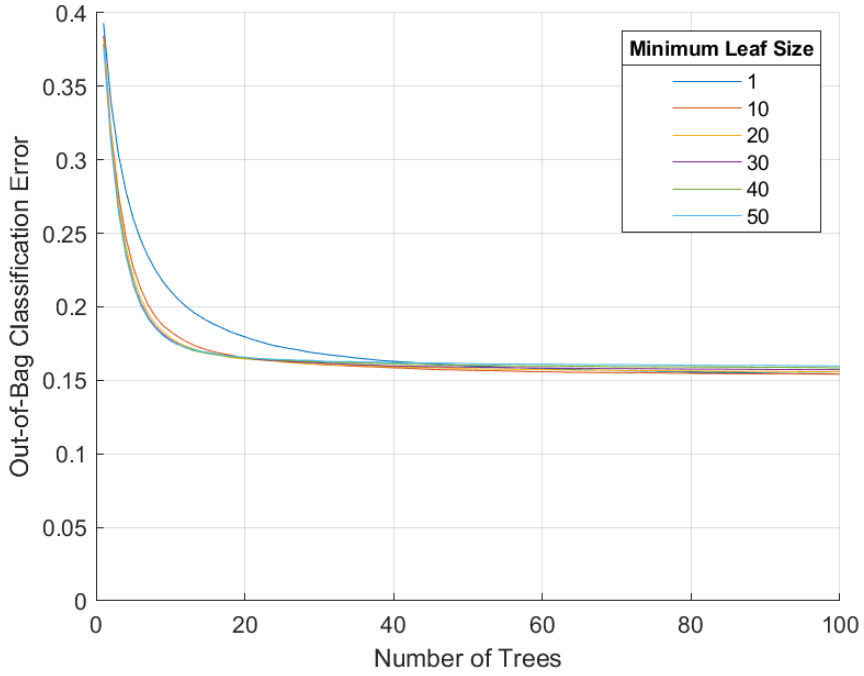


Figure 25. Out-of-bag classification error for random forests of various minimum leaf sizes

No significant improvement in accuracy is shown beyond 50 trees for all MLS. The classification error for all forest sizes appears to asymptotically approach a value near 0.15 as the number of trees increases. The results indicate that after 40 trees there is only a nominal reduction in classification error due to MLS, and that there appears to be an intersection of sorts at 20 trees. At 20 trees all MLSs except 1, 5, and 55, have a classification error of 0.165 ± 0.001. From this analysis, it was decided to constrain the search for an acceptable forest composition to a size of 20 to 50 trees.

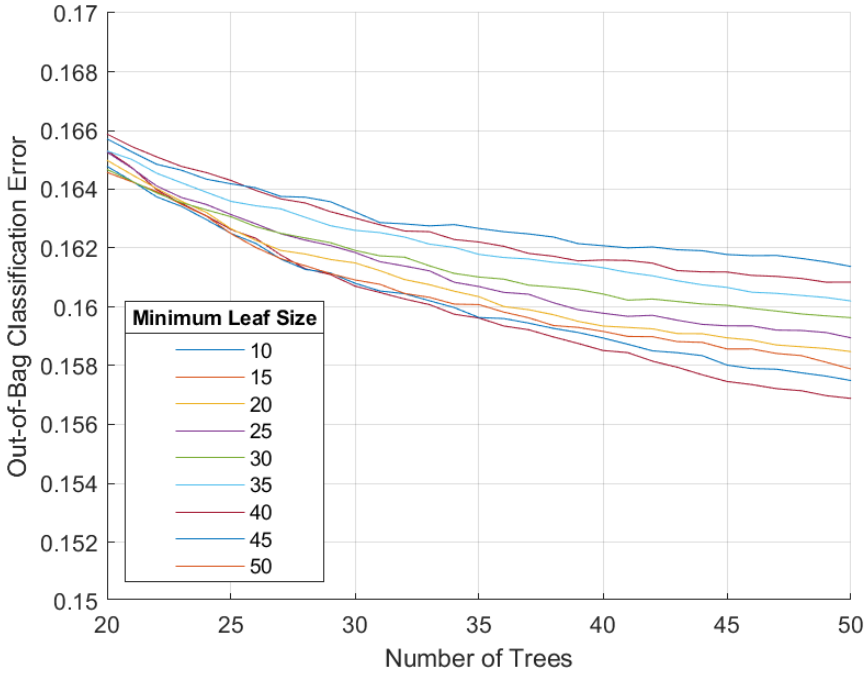


Figure 26. Enhanced view of the out-of-bag classification error

Analyzing all MLSs tested except 1, 5, and 55 in this range of 20–50 trees, illustrates that there is a small, approximately 0.01, improvement in classification error for any MLS, which is shown in Figure 26. An improvement of this magnitude is considered insignificant as the classification results will be filtered to remove misclassifications. Tests conducted with respect to processing time revealed that a model with 20 trees and MLS of 10 provided adequately fast processing time. Whereas tests conducted with 50 trees were not fast enough to prevent lurching motion in the AGV.

The forest with 20 trees and MLS of 10 was not the only forest composition that met the processing time requirements, but was settled on early in this research to move forward in developing the rest of the navigation solution. The testing of random forests was limited due to testing forests with different features that were ultimately deemed unsuitable for the task. Two examples of these are the use of SURF features and pixel intensities with color representation. SURF features were disqualified due to the long processing time

associated with extracting them from the interest points. While pixel intensities with color representation was disqualified after egregious terrain misclassifications during a test in the workspace, even though it had outperformed the pixel intensity versions in laboratory tests with mixed terrain images.

The selected forest, with 20 trees and MLS of 10, was tested with images containing mixed terrain types to analyze effectiveness, identify issues, and to analyze how the filtering process effected the classification results. The random forest implemented on the AGV is compared to the random forest with 100 trees and MLS of 5 as it had the least classification error of any developed during the initial assessment of forest composition. The classification error of the implemented forest is 0.1653 compared to the 0.1531 for the forest with 100 trees and MLS equal to 5. Going forward these forests will be referred to as the Implemented Forest and the Least-Error Forest.

Comparing the performance of the two forests involved generating predictions for the same image, and overlaying the results on the image. The overlay uses red X's to indicate classification of hazardous terrain, and green O's to indicate classification of traversable terrain. Comparing the markers to the terrain they overlay allows analyzing the effectiveness of the terrain classification. For example, if a green O overlaps traversable terrain then the terrain has been correctly classified, and if a green O overlaps hazardous terrain then the terrain has been misclassified. Similarly, red X's are intended to overlay hazardous terrain, and not traversable terrain.

An example of prediction results for an image containing both grass and concrete is shown in Figure 27. Example prediction results for an image containing both mulch and sand paths is shown in Figure 28.

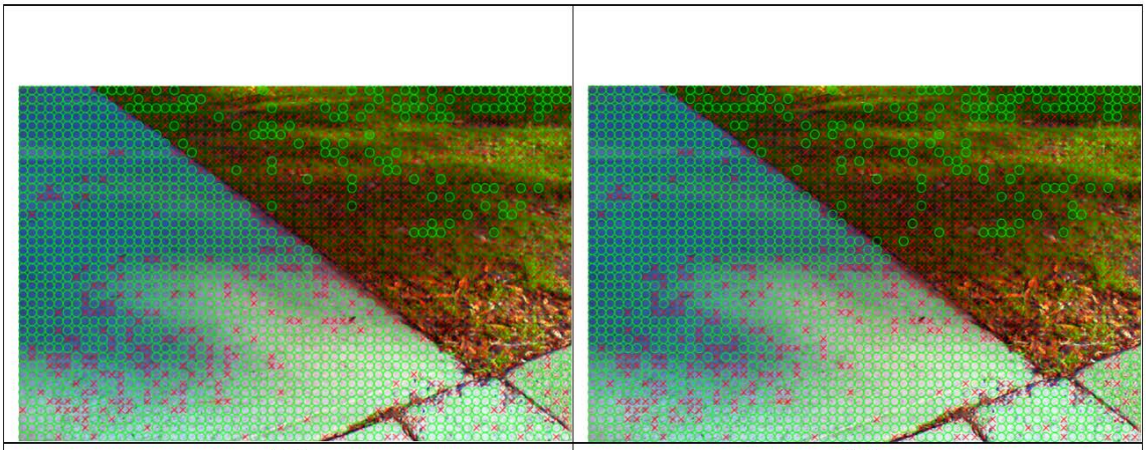


Figure 27. Visualized prediction results for an image with concrete and grass from (a) Least-Error Forest, and (b) Implemented Forest

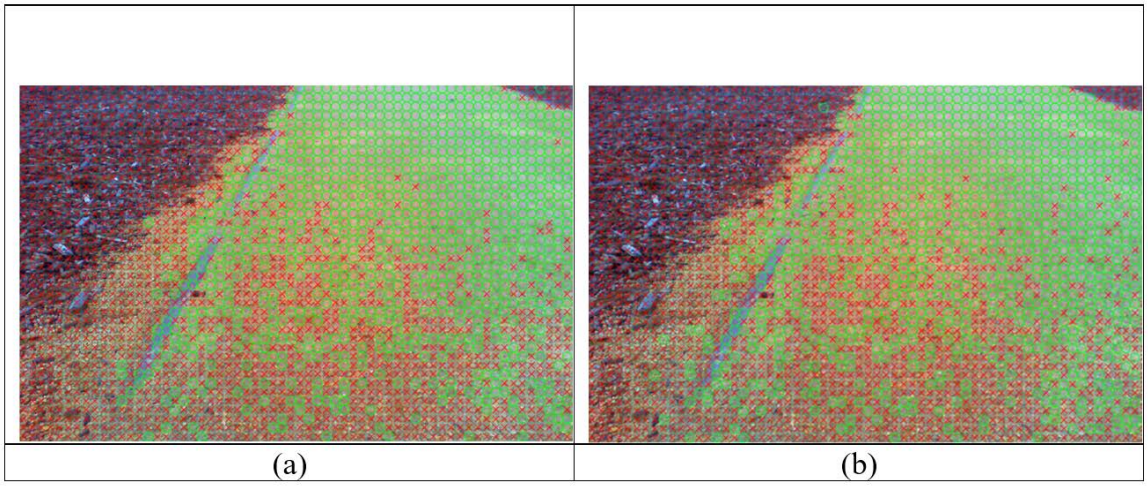


Figure 28. Visualized prediction results for an image with mulch and sand path from (a) Least-Error Forest, and (b) Implemented Forest

The visualizations shown in Figures 27 and 28 provide two insights from this process. First, the Least-Error Forest does perform better on the test images, but the difference is not staggering. Second, false positives are an issue for all types of terrain analyzed; however, it was much more rampant when the actual terrain was the sand paths. In all images analyzed in this way, the forests were least successful in properly classifying sand paths. These images set the basis for a comparison of the original, noisy classification, to the filtered version.

The filtering process is intended to remove isolated false classifications and small regions of falsely identified hazardous terrain from *PredMat*. Results of filtering the *PredMat* visualizations shown in Figures 27 and 28 are visualized in Figures 29 and 30, respectively.

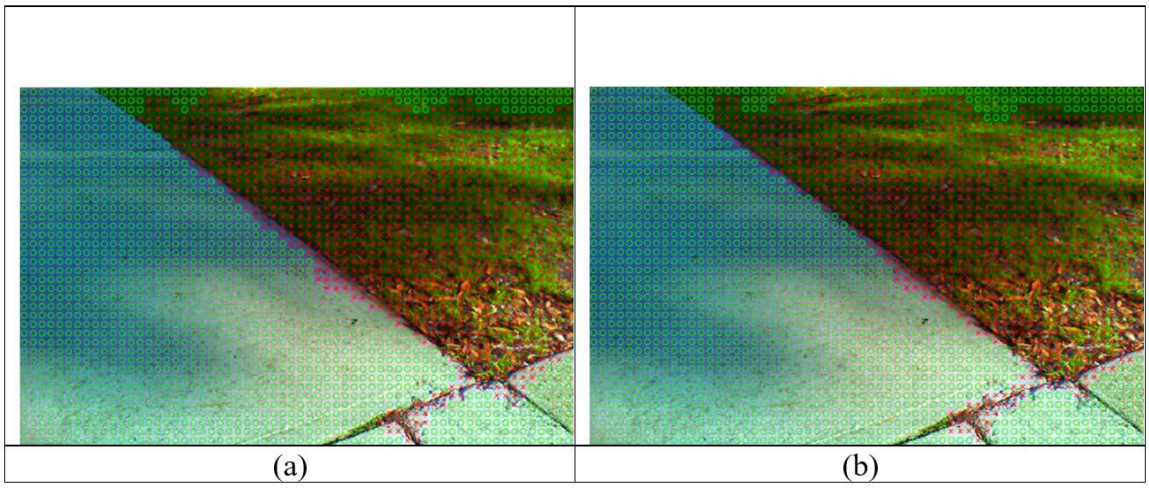


Figure 29. Visualized filtered results for an image with concrete and grass from (a) Least-Error Forest, and (b) Implemented Forest

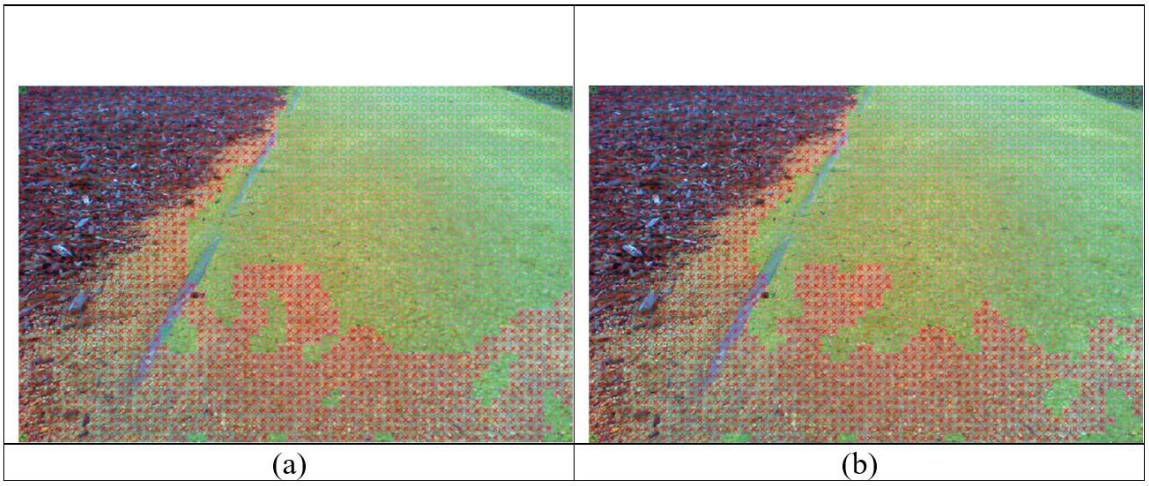


Figure 30. Visualization of the filtered results for an image with mulch and sand path from (a) Least-Error Forest, and (b) Implemented Forest

One of the immediate observations from Figures 29 and 30 is the remaining misclassification of sand paths. This was observed throughout numerous images with sand paths, even after the filtering. This was a limiting factor for the terrain classification. To handle this limitation, the terrain force gain was tuned to ensure the system can maneuver over sand paths and still avoid truly hazardous terrain.

Overall, the filtering process accomplishes the desired goal of removing small regions of misclassified terrain as represented in Figures 29 and 30. The result of the filtering process shows very similar final classification results regardless of the original random forest model. The Least-Error Forest is still slightly more accurate, but given the efficient processing time of employing the smaller forest, the trade in accuracy is considered worthwhile for this work. The tendency for the random forest to misclassify sand paths rolls over into the results from filtering. This method of effectively classifying terrain allowed developing a repulsive force for the artificial potential fields algorithm and for storing locations of hazardous terrain in the AGV databanks.

B. MEMORY AUGMENTATION

One of the fundamental elements used in developing the navigation solution was the storing of sites that generated local minima. This process is described in Chapter IV. By the end of this thesis work, the AGV had 148 sites stored as obstacles. The stored obstacles had been periodically reviewed to remove those added due to terrain misclassification and significant localization errors. Obstacles are shown as red diamonds overlaid on imagery of the workspace from Google Earth Pro in Figure 31.

Figure 31. Locations of obstacles identified by the AGV. Adapted from [37].

The system was originally run by using the locations of previously encountered obstacles to develop a repulsive force to avoid them, and this led to odd behavior from the AGV. This was especially true when there were multiple known obstacles in an area. To investigate why this was happening, the situation was simulated in MATLAB to determine methods to improve the performance as described in Section IV.D. The results of those simulations are discussed in the next section.

C. SIMULATION RESULTS

The initial simulation was conducted with the known obstacles defining the edge of the region in their true position to determine a performance baseline. An example of the initial simulation results are shown in Figure 32, where the black markers indicate the motion of the robot under artificial potential fields, and the magenta squares indicate the local minima escape mode. This simulation resulted in the simulated robot always being able to maneuver from the initial position to the goal without issue, and only a small portion of the trajectory was spent in the local-minima-escape mode.

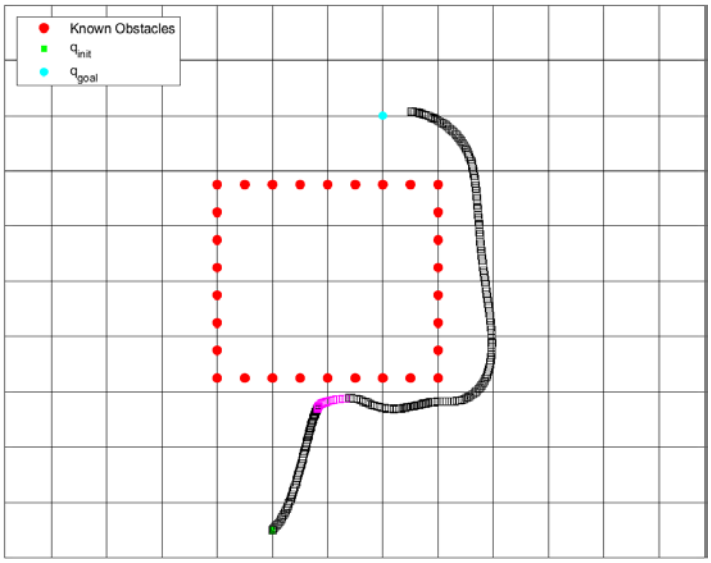


Figure 32. Simulation of the known obstacle effects with true position of known obstacles

The position of obstacles stored by the AGV were expected to be noisy due to the localization error caused by the GNSS/INS. To test the effects of known obstacles with noisy positions, random noise was added to the positions of the simulated obstacles, and the simulation was run several times. Examples of the results are shown in Figure 33. The results were used as a qualitative means of assessing the performance of the system and the representation of obstacles.

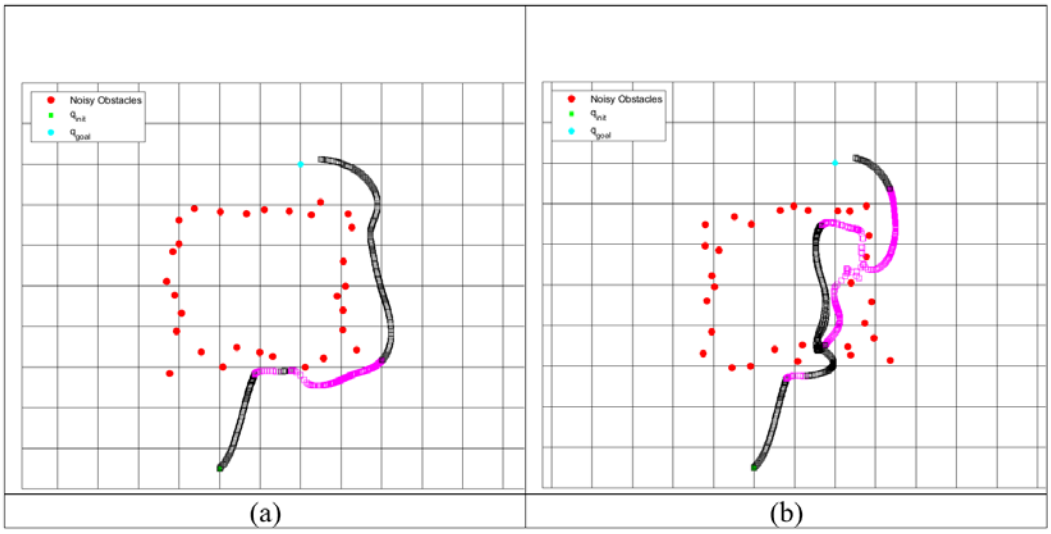


Figure 33. Example simulation results of the robot trajectory in an environment with known obstacles having noisy positions

Results like those shown in Figure 33(a) visualize how noisy positions effect the route and behavior of the robot, and most notably tend to cause the system to spend more time escaping from local minima instead of flowing to the goal.

The results shown in Figure 33(b) are not realistic as it shows the simulated robot entering the obstacle region defined by the points. In an actual trial the sensors on the AGV would prevent this from occurring. Preventing this, without relying on sensors, is handled by manipulating how close together obstacles are allowed to be stored. This distance could be set to zero, but this would be inefficient to store and would provide many sources for the repulsive force which could bog down the system. The path planning algorithm that was implemented later led to the decision to require known obstacles be at least one meter apart. This makes sense as obstacles within one meter of one another were logically part of the same obstacle/obstacle region. This eventually ensured paths were planned around closely spaced obstacles and prevented the AGV from slipping through closely spaced obstacles. As mentioned in Section IV.C.1.a, the noisy positions of obstacles negatively affected the AGV performance. To refine the perceived location of obstacles, a least squares fit is conducted over a finite neighborhood.

The least squares fit uses a four by four-unit neighborhood. Results of the simulated robot after the least squares fit was conducted are shown in Figure 34, where the refined obstacle positions are shown as blue squares, and all other symbology remains the same.

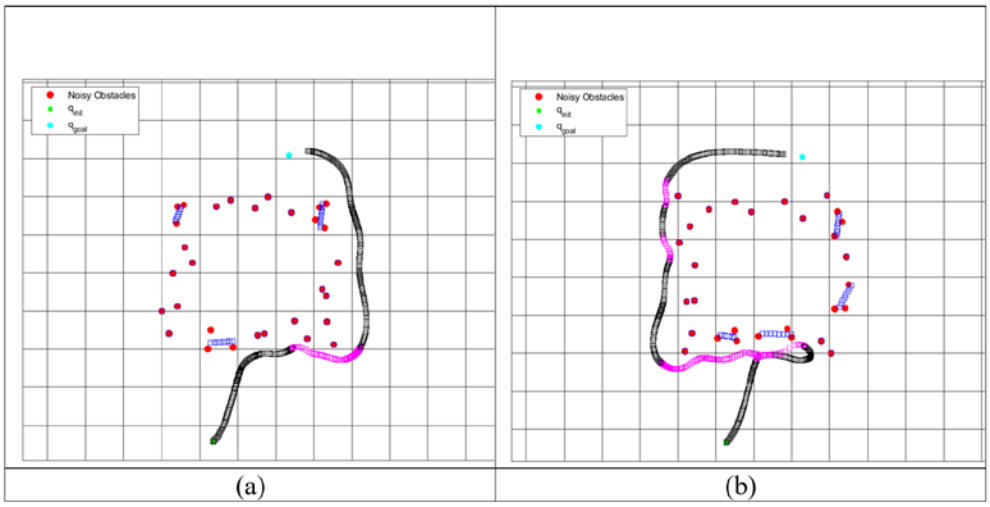


Figure 34. Example simulation results of the robot trajectory after refining noisy obstacle positions

The simulations show that the least squares fit works to clean boundaries defined by noisy positions along that boundary, but it does not necessarily match the orientation of the boundary. It also shows that it does not necessarily improve the performance of the system, which is still largely impacted by the geometry of a few of the simulated obstacles.

The implementation of a path planning algorithm was a logical extension after developing a workspace representation to refine obstacle positions. Implementing the same approximate cell decomposition method, but with one-unit by one-unit cells the A* search algorithm, *AlphaStar.m*, was then employed to find an optimal path from the initial position to the goal. This was executed in simulation several times to ensure valid paths were planned and executed. A representative result is shown in Figure 35.

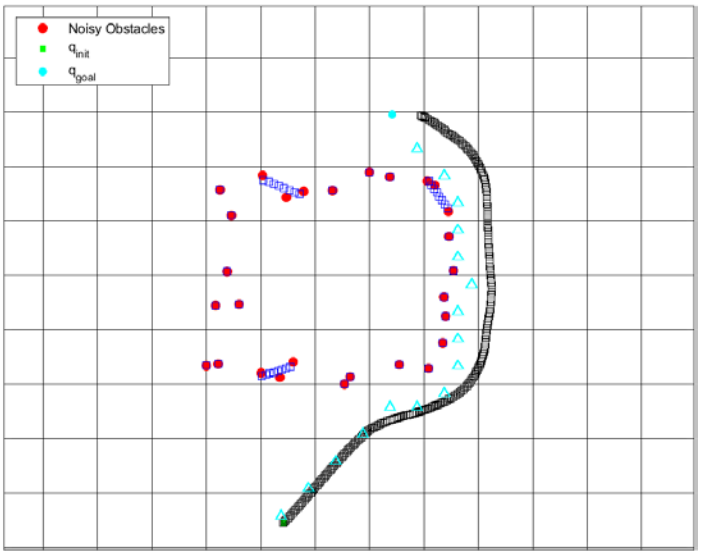


Figure 35. Simulation results from planned path trajectory for the robot with refined noisy obstacle positions

The results from the simulations involving planned paths have little difference from one another. The simulated path was planned around the right side of the rectangular region, and the simulated system did not enter the local minima escape mode. This demonstrated that the path planning algorithm worked for this configuration. Consequently, it was deemed ready for employment on the AGV. While further simulation of the path planning algorithm could have been conducted to test for limitations or issues

with the path planning algorithm or the workspace representation, it was decided these were unnecessary. This is based on the assumption that the real-world testing would reveal any issues that may exist with either.

D. NAVIGATION IN A REAL-WORLD ENVIRONMENT

The next step was to employ the AGV, equipped with the fully integrated navigation solution, in a real-world environment. Numerous trials were conducted at the Naval Postgraduate School campus. Each trial required establishing a local network and accessing the AGV via secure shell. From this interface, a ROS network was launched before the trial could be started from MATLAB. Not every trial resulted in the AGV successfully navigating from the initial position to the goal. Several trials are represented and discussed throughout this section in regard to the effectiveness of the navigation solution in the operating environment.

1. Example 1: Operations on Tiled Concrete

This example consists of the robot traveling from its starting point to a nearby initial goal, before having to travel around a region of hazardous terrain to the final goal. The results of the trial are shown in Figure 36. The white line represents the route of the robot according to the filtered GNSS/INS data, from the ROS /geonav transform, and the blue line is the route recorded from the GNSS fix. The differences are not significant, but are shown as the localization of the robot proved to be the most significant issue during the trials. The red diamonds represent the obstacles the AGV had stored in its memory.

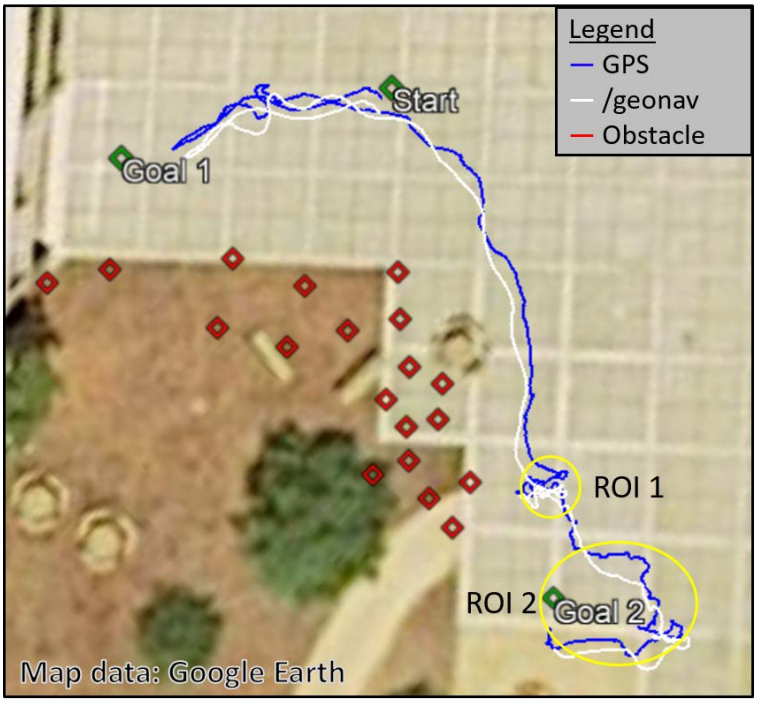


Figure 36. Navigation on tiled concrete. Adapted from [37].

The trial shown in Figure 36 was the result of the system planning a route after having interacted with the workspace and storing the locations of several sites of hazardous terrain. This trial demonstrated a few successes, primarily that the navigation solution works by successfully planning and executing a path from the start point to the goals. This leans on the AGV successfully identifying and storing the location of hazardous terrain. The trial is not without issue in that the terrain classification experienced issues when the AGV reached the first region of interest, ROI 1. When the AGV reached ROI 1, it reacted to terrain it had misclassified as hazardous. This caused it to turn away from the waypoint it was headed toward, and then turn back toward it before continuing the planned route. The portion of the path in ROI 2 is also the result of terrain misclassification. In this case, the misclassification triggered the AGV to enter the Terrain Following state several times. Upon exiting the Terrain Following state, the system arrived within range of the goal to complete the trial. The run resulted in several improperly placed obstacles that were subsequently removed from the AGV memory. The issues with classifying terrain in this trial are attributed to rains that had occurred in the days preceding the trial. Given the terrain classifier was trained on data gathered during sunny days with no precipitation prior to data

collection, any alterations could feasibly affect the ability of the algorithm to accurately classify terrain.

2. Example 2: Operations on Sand Paths

Example 1 demonstrated the AGV was able to plan and execute a path along tiled concrete with few instances of terrain misclassification. The laboratory testing, discussed in Sections V.A, identified that the terrain classifier was prone to misclassifying sand paths. To test the performance of the system on sand paths in the real world, a trial was devised that required the AGV to travel along sand paths to effectively reach its goal.

The test required that a path be planned and executed around a portion of mulch and bushes/trees that block a straight-line path from the first goal to the second. Prior to the trial shown in Figure 37, the robot was run three times in this area. One of those reached the goal, and two were stopped due to localization issues. The localization issues caused the AGV to move as if it were in a different portion of the workspace and to store obstacle positions in locations that were inaccurate. These obstacles were removed from the AGV databank, and the runs are omitted.

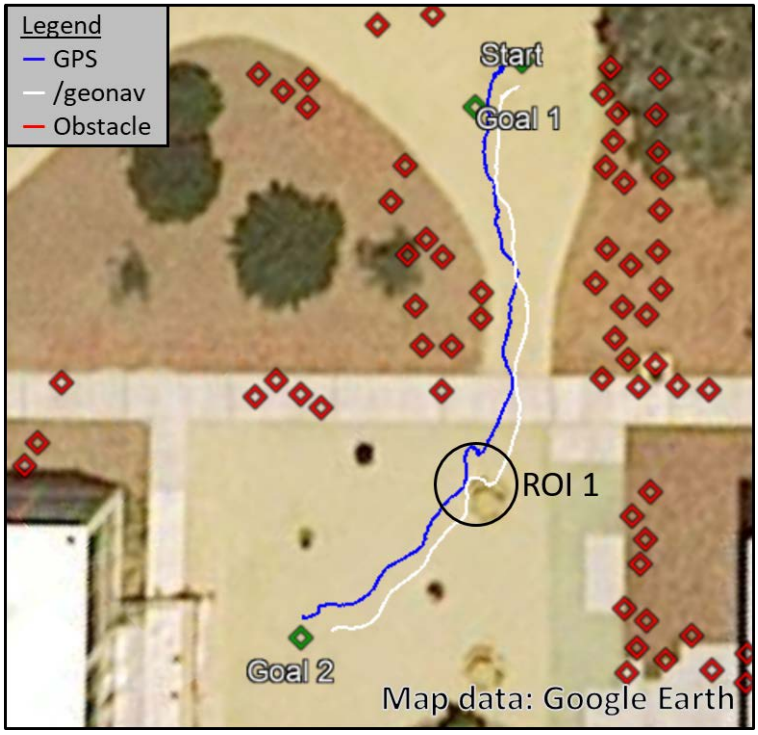


Figure 37. Navigation on sand path. Adapted from [37].

This path successfully navigates from the start point to the desired goals. The small deviation of the AGV trajectory in ROI 1 is the result of the AGV altering its course to avoid a picnic table in the area, close to where the picnic table is in the image, which is desired behavior for the AGV.

3. Example 3: Long-Range Issues

Upon completing shorter range trials, it was decided to conduct longer range trials to ensure the system was able to build and execute a navigation solution over a larger portion of the workspace. During this effort it was discovered there were two phenomena that are worth discussing. First, a trial in which the AGV gets trapped due to several factors including geometry, the physical properties of the environment, and ultimately battery life. Second, a trial in which the proximity of known obstacles and the repulsive force due to known obstacles prevents the system from being able to execute its planned path.

In Figure 38, the AGV path is shown heading right and up in the image, but then as the path comes to the mulch it then follows along the mulch toward the bottom of the image. This was due to how the AGV approached the hazardous terrain. The issue with this run starts when the AGV begins to move toward the right after moving toward the bottom of the image. At this point the AGV is trying to avoid the mulch that is on one side, and must avoid the door of the building it is heading toward. The AGV entered and stayed in the Terrain Following state, identifying several areas of hazardous terrain, in an attempt to escape this situation, but the AGV ran out of battery power before it was able to escape the nook it found itself in.

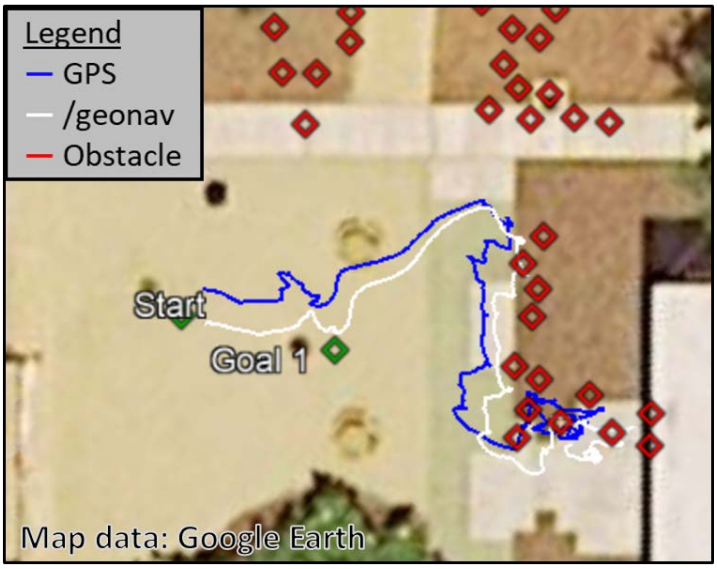


Figure 38. AGV trapped by unique circumstances. Adapted from [37].

The trial shown in Figure 39 shows the path of the robot moving toward the second goal, which is located high and right off the image. As it approached the region of mulch it enters the Terrain Following state at ROI 1. Note that while the path in Figure 39 shows the AGV travelled through the mulch, the AGV never had more than one wheel in the mulch throughout the trial. As it travelled along the mulch the AGV tended to oscillate along the border of the traversable and hazardous terrain due to the limited field of view of the camera. While in this mode it added hazardous terrain locations to its memory as it moved along the border of the mulch. The AGV eventually escaped Terrain Following mode, and was turned around by a bench located at ROI 2. Once the AGV was turned around it re-entered the Terrain Following state, and headed back toward the point it initially entered the Terrain Following state, and escaped the Terrain Following state near that point. The system then planned a path that started the AGV down the sidewalk, but was it not able to navigate down the sidewalk due to the force from the known obstacles. The AGV became stuck around ROI 3 as it was trapped in a local minimum due to the known obstacle force. The trial was cut short at that point. To overcome situations such as this, the gain of the known obstacle force must be tuned to allow maneuvering through gaps along sidewalks, but prevent passing through narrower gaps, or by devising an escape mode for this phenomenon. Another takeaway from this trial is the random drastic

localization errors that occured. An example of these is the lines from the /geonav transform that are shown in ROI 4. These lines extend out several kilometers from the true position, and shortly afterwards the localization returns to the approximate position of the AGV. These short duration jumps tend not to affect the overall trials but do cause issues during the time the localization is drastically off. These can cause the entire behavior of the AGV to change. This is to be expected as it is acting on the data it has from its sensors while continuing to seek out the prescribed goal.

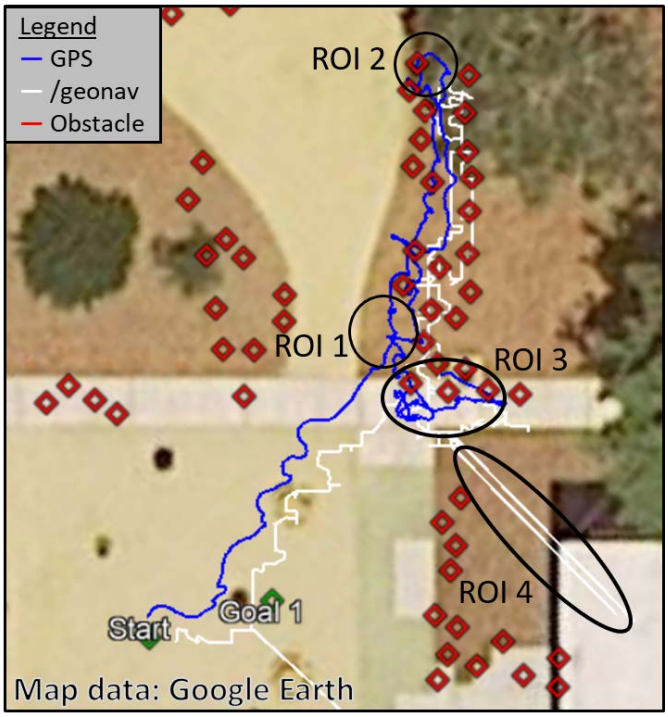


Figure 39. Failed long-distance trial due to known obstacle force. Adapted from [37].

4. Example 4: Long-Range Improvements

The trials shown in Figures 38 and 39 highlight several of the limitations of the AGV. In contrast to this is the trial shown in Figure 40, in which the system can plan its routes to the goals, avoid hazardous terrain, and overcome complex obstacle configurations.

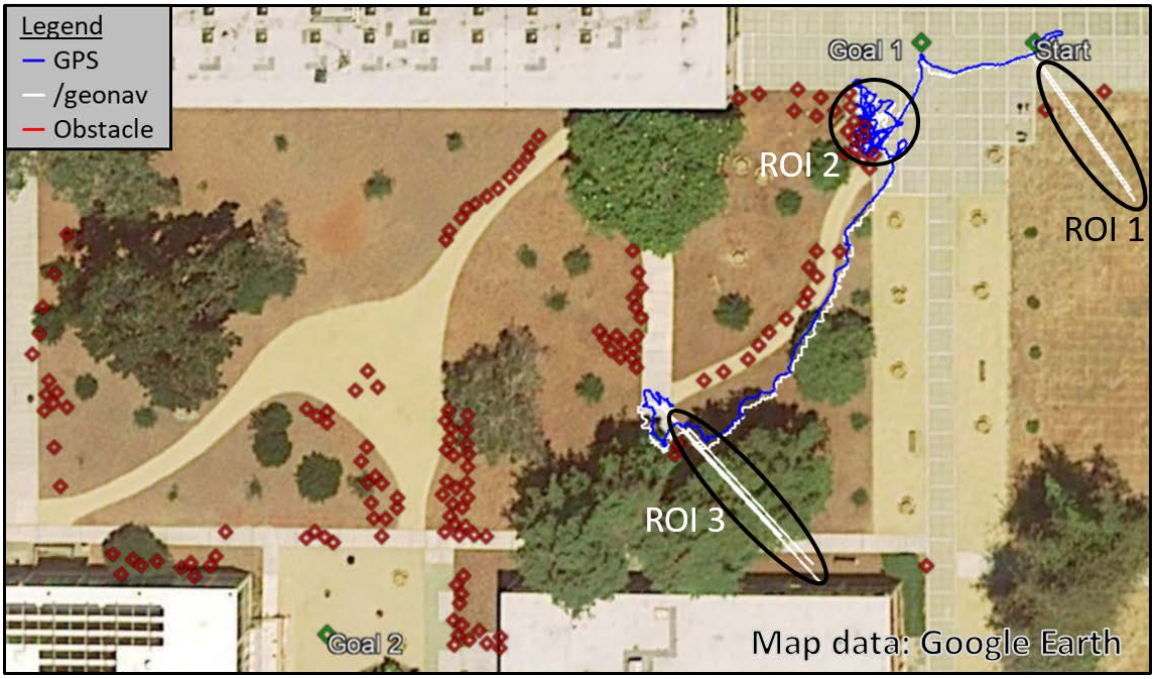


Figure 40. Long-range trial with improved performance. Adapted from [37].

The information in Figure 40 is misleading in that it shows the AGV travelling in the mulch, this is due to the localization error. The general path is correct in that the AGV started on the tiled concrete, it then planned a path to the first goal and reached it without issue. From there the path planner developed a route to the second goal that led into the mulch. The portion marked as ROI 2 shows the path of the AGV as it entered and exited the Terrain Following state trying to escape this region. After the AGV identified there was not an available path in that area it then moved to the sand path. It travelled along this path, and only entered the Terrain Following state once before coming to the concrete walkway. It was at this point that the AGV battery was so depleted that the system stopped, and the trial was ended. Additionally, several instances of drastically inaccurate localization, shown as ROI 1 and ROI 3 were experienced in the trial. Again, these were not typically able to derail a trial, but do hinder the performance for a short period of time.

The tests of the system were designed to examine the effectiveness of each element of the designed solution. Laboratory tests of the classification process provided a method of determining an efficient baseline classifier. They also allowed determining that the filtering process improved the classification results of the implemented random forest to a

point that was comparable with the best forest grown. The simulations allowed developing a method to mitigate the effect of noise in the position of obstacles, and the path planning method that allowed for efficient avoidance of known obstacles. Real-world tests demonstrated the effectiveness of the implemented solution under several different scenarios and environments.

The tests also revealed limitations of the current system. The most notable limitations include the narrow field of view of the camera, the accuracy of terrain classification, and the effects of imprecise localization. Each of these limitations degraded the performance of the system and made it less effective in navigating through the workspace. The limitations did not prevent the system from accomplishing the desired goals of this thesis. The limitations and the assessment of the goals of the thesis are discussed in Chapter VI along with potential future work.

THIS PAGE INTENTIONALLY LEFT BLANK

VI. CONCLUSION

This thesis research developed a system that is capable of avoiding hazardous terrain. This same system also learns about its environment and uses that information to perform more effectively in future excursions. This was accomplished by pairing an efficient terrain classification algorithm with a navigation solution that accounts for prior interaction with the workspace. The navigation solution employs the motion planning of artificial potential fields and conducts path planning using the *AlphaStar.m* script. *AlphaStar.m* uses a modified application of approximate cell decomposition and A* search to find the optimal path from the initial position of the AGV to the goal. An assessment of the thesis goals, discussion of the limitations, and areas for potential future work are contained within this chapter.

A. ASSESSMENT OF GOALS

The purpose of this thesis work was to develop a method for an autonomous ground vehicle to identify and avoid hazardous terrain. Two goals were developed to achieve this purpose. The first goal was to develop a machine learning algorithm to classify terrain. The second goal was to develop a method for avoiding hazardous terrain once it had been identified. The assessment of these goals is discussed in this section.

The goal of identifying terrain types using a machine learning algorithm was accomplished by growing a random forest that was trained using images of homogenous terrain from the testing area. The forest selected for this thesis research was chosen from the tradespace of predictive accuracy and processing speed, and it provides adequate predictive accuracy in a timely manner.

The second goal—avoiding hazardous terrain once it had been identified—was accomplished using a navigation solution, coupling motion planning with path planning. The motion planning method used in this thesis augmented a previously developed artificial potential fields algorithm to account for hazardous terrain classified by the random forest as well as locations where the AGV had previously identified points of interest in the workspace. The path planning methods developed account for noisy

locations of known obstacles, and they use approximate cell decomposition along with the A* search algorithm to find an optimal route to the goal with the information it has about the workspace.

The solution enabled the AGV to identify and avoid hazardous terrain during normal daytime operations. This in turn allowed the AGV to independently learn about its environment. By learning about its environment and equipping the AGV with a path planning solution, the AGV was able to plan its routes to avoid areas with known obstacles, without user input. The solution devised is not without exception and limitation, the next section explores some of the limitations identified during the thesis research.

B. LIMITATIONS

Limitations exist in several elements of the solution devised for this thesis research. The limitations identified will be briefly discussed in this section including the narrow field of view of the camera, the predictive capabilities of the random forest, and the localization method used.

The decision to use a monocular vision solution resulted in a narrow field of view that limited the ability to visually observe the robot surroundings. Using a camera with a wider field of view or multiple cameras would allow for improved observation of the robot surroundings. In the context of this research there were numerous times throughout the real-world testing that the AGV would turn to avoid terrain, and as soon as the terrain left the field of view it would forget the terrain was there. This resulted in the AGV turning back toward the terrain almost immediately after the terrain was out of the field of view. A wider field of view would still have this issue, but with a wider field of view the system would be better able to travel along a terrain feature, without oscillating, while moving toward its goal. An alternate method of handling this issue would be implementing a short-term memory on the AGV that allowed it to remember and react to obstacles within a small radius of it.

The random forest developed had an error of approximately 16%, according to laboratory tests. It was noticed during real-world testing that it consistently struggled with properly classifying sand paths and even had issues classifying terrain if the environment

did not match the environment the training data was collected in. This limits the ability of the system to maneuver throughout the workspace, and at times required the user to remove obstacles the system erroneously identified along otherwise traversable terrain. It is expected that by using the proper combination of feature, machine learning algorithm, and parameters for the machine learning algorithm it is possible to improve prediction accuracy and speed which would allow the system to navigate better and more efficiently. This must also be paired with a set of training data that includes examples from numerous environmental conditions to build a more robust classifier.

The localization of the AGV caused issues throughout testing and frustrated the use of known obstacles for path planning and known obstacle avoidance. The localization solution was taken from the GNSS/INS. The GNSS/INS unit has built in filtering and optimal estimation capabilities. The optimal estimation process onboard the unit has yet to be fully analyzed in the theses that have used the similar AGV design. The resulting localization of the AGV is accurate, though not precise. Due to the positional error shifting with each location update, the noise in the system becomes problematic. The major problems and limitations arise when the system cannot reach the desired location due to the localization error. The localization errors associated with the AGV are highly prohibitive as they affect the ability of the system to navigate to features that will require precision such as doorways, ramps, and elevators. The limitations and problems encountered during the thesis work inspire the concepts of future work for this project.

C. FUTURE WORK

The areas for potential future work were conceived due to the limitations observed during testing and evaluation of the current thesis research. Some of the areas include the terrain classification process, the sensors used, and the localization methods.

The terrain classification process was approached using the random forest machine learning algorithm, but there are many other machine learning algorithms that could have been used. These may provide better results with the data that is available from the sensors onboard the system. Additionally, the actual features used in training and classification may be investigated to determine if the use of a different feature may provide better

classification. Work may also be done to classify terrain according to the subcategories captured in this work rather than the just the two categories used. Multiple categories could be used for motion planning, choosing routes through terrain that is best for the AGV, and providing greater detail to the information the AGV has about the workspace.

Research opportunities exist in improving the random forest developed to classifying terrain. This could include exploring improving the classification accuracy through feature type or size and composition of the forest. It may also work to reduce classifying traversable terrain as hazardous by incentivizing the classification of traversable terrain during the random forest growth process. Additionally, work could be done to reduce the depth of trees in the random forest to provide faster classification. Work of this type would require further investigation of the tradespace balancing classification accuracy and processing speed.

Another element of the overall classification process that could be explored is the filtering process used to eliminate false classification. This could explore other morphological processes, other image processing techniques, or it could account for information from the image in filtering the classification results from the machine learning algorithm.

Augmenting the sensor suite of the system would serve a plethora of purposes. By including a LIDAR capable of providing information about the ground around the AGV it may be possible to improve the terrain classification results. A LIDAR capable of this would also provide information about obstacles that are below the horizontally mounted LIDAR which present a rare but significant issue for the AGV. Whereas a second camera could be installed to widen the field of view and give more information about the workspace if incorporated with the terrain classification process.

The current method of localization relies on the use of the GNSS/INS which has demonstrated to be useful but riddled with error large enough to affect system performance. Exploring methods of improving the localization effort could include using the raw data from the INS, GNSS, the wheel encoders, and the camera to develop an optimal estimation to improve the localization. Alternatively, developing a map of the workspace with the

LIDAR, and then using the LIDAR readings to correlate to that map is another method of potentially improving the localization. An additional localization methodology worthy of exploration could be using time delay of arrival of signals emitted from known locations to identify the location of the AGV.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX A. DATA COLLECTION SCRIPT

```
%% Matlab Script
% Grab webcam image and save to a file while iteratively naming them

% If the path is not added and saved due to permissions the below is
% required
% addpath
/home/calvin/Documents/MATLAB/SupportPackages/R2016b/toolbox/matlab/web
cam/supportpackages

clear cam
cam = webcam('Microsoft');  % camera object is set to the 'Microsoft'
% camera attached, may need revision if using multiple cameras with the
% same maker

I = snapshot(cam);  % grabs a snapshot from the webcam
for i = 1:5         % this loop is added at the front end of the script
    % due to observed adjustments to the cameras digital parameters
    % that effect contrast, saturation, etc.  The camera adjusts those
    % automatically, and by taking a series of pictures they appear to
    % settle so the images are no longer observably inaccurate.
    I = snapshot(cam);
    imshow(I)
    pause(2)
    close all  % none of these photos are saved
end
%% Iterative loop to grab a photo every 5 seconds
i = 1;  % this counts the iterations/number and name of the photo

while i > 0

        if i == 1   % make a directory for the photos to be saved in
            dirName = 'Gathered_Images';
            mkdir(dirName);        % redundant for i > 1
        end

        ImDir =
sprintf('/home/calvin/pioneernav/MATLABScripts/%s',dirName);
        % sets the filepath for the newly created directory
%% Grab the webcam snapshot and save to file

        I = snapshot(cam);  % grab photo

        fileName = sprintf('Img%03d.png',i);  % write image file name
        FullFileName = fullfile(ImDir,fileName); % set fullfile name
        imwrite(I, FullFileName);  % write image, file name ensures
format
        pause(5)  % wait 5 seconds, can be adjusted

        i = i+1;  % increment the index
end
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX B. GROWING A RANDOM FOREST

```
%% Build Tree Classifier

load('TerraVars.mat', 'FixPts');

good = zeros(1,146);
bad = ones(1,135);
GrdTruth = [good, bad];

% Mix the photos (attempt to avoid biasing)
P = randperm(281);
PermTruth = GrdTruth(P);

charVals = zeros(719360,122);
ImDir = sprintf('/home/calvin/pioneernav/MATLABScripts/Train_Img');


LL = sub2ind([1280,800], FixPts(:,1), FixPts(:,2));
for j = 1:281  % Number of images to analyze
    % Load Image
    filename = sprintf('Img%03d.png',P(j));
    FullFileName = fullfile(ImDir, filename);
    I = imread(FullFileName);
    I = rgb2gray(I);

    [features,~] = extractFeatures(I,FixPts,...
            'Method', 'Block', 'BlockSize', 11);

    features = im2double(features);


    charVals((1+(j-1)*2560):j*2560,:) = ...
    [features, PermTruth(j)*ones(2560,1)];
end

%% Build the model
TerrainModel = TreeBagger(20, charVals(:,1:121), charVals(:,122),...
    'Method', 'classification', 'minLeafSize', 10);

save('TerrainModel.mat','TerrainModel');
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX C. MISSION COMMAND

```
%% Mission Command
% code adapted from Matt Audette's thesis code
% This code allows the user to either load in a set of pregenerated
% waypoints for execution or write them in directly.  It then calls the
% state-based machine to drive the AGV to those points.
flag = false;

if flag                         % load a pregen coordinateList
    load('Experiment8WS')
else                            % write coordinateList
    coordinateList = [36.595276, -121.875642,0;
                      36.594986, -121.875573,0];
end

% Loop the State-Based Machine Code:
for i = 1:size(coordinateList, 1)
    %Send the coordinate and waypoint #
    %and wait for the robot to go to that point:
    potentialFieldToWaypointTerrainFollowWorking( coordinateList(i, :),
i)
end
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX D. STATE BASED MACHINE

```
%% Potential Field to Waypoint Function
% Caliph Lebrun

% This is function an evolution of work done by Calvin Hargadine and
% Matthew Audette.  This script subscribes and publishes to ROS nodes
to
% gather information about the AGV's surrundings and then pursue its
goal.
% It uses a state-based machine approach to automate a P3-AT.  This
allows
% it to plan paths, execute those paths via artificial potential
fields,
% escape local minima, and perform emergency avoidance of dynamic
% obstacles.  Recent additions, from this work, allow it to store
location
% information about terrain and other local minima for future path
% planning.  The current sensor package includes GNSS/INS, LIDAR, and
% vision.  This function provides the execution needed by the
% MissionControl.m script.

function StateBasedExecution(coordinates, goalnum )
    %%%% ENSURE ROS MASTER NODE IS STARTED AND MATLAB NODE GENERATED
PRIOR TO
    %%%% RUNNING THIS SCRIPT -- USE rosinit

    %% Setup and parameter initialization

    % Create global variables for use in communicating with ROS system
    global Pose
    global Laser
    global Goal
    global NavStatus
    global GPSFix

    % Create ROS publishers, subscribers, and service client
    poseSub = rossubscriber('/geonav_p3odom',@p3atPoseCallback);
    laserSub = rossubscriber('/scan',@p3atLaserCallback);
    cmdPub =
rospublisher('/RosAria_Node/cmd_vel','geometry_msgs/Twist');
    goalPub = rospublisher('/nav/goal_odom','nav_msgs/Odometry');
    casePub = rospublisher('/current_case','std_msgs/String');
    goalSub = rossubscriber('/geonav_goalodom',@p3atGoalCallback);
    navstatusSub = rossubscriber('/nav/status',@p3atNavStatusCallback);
    fixSub = rossubscriber('/gps/fix',@p3atGPSFixCallback);
    client = rossvcclient('/reset_kf');

    % Pause for publisher/subscriber registration
    pause(2)

    % Create empty messages for publication
    caseMsg = rosmessage(casePub);
```

```
    cmdMsg = rosmessage(cmdPub);
    goalMsg = rosmessage(goalPub);

    % Get parameters and goal information the robot
    [param, ~] = robotConfigReader_multigoal;

    % Ask user for desired goal number
    current_goal = coordinates;

    % Publish initial goal message for ROS system transform
    for k = 1:5
        goalMsg.Pose.Pose.Position.X = current_goal(2);
        goalMsg.Pose.Pose.Position.Y = current_goal(1);
        goalMsg.Pose.Pose.Orientation.X = 0;
        goalMsg.Pose.Pose.Orientation.Y = 0;
        goalMsg.Pose.Pose.Orientation.Z = 0;
        goalMsg.Pose.Pose.Orientation.W = 1;
        send(goalPub,goalMsg);
        pause(0.1)
    end

    % Get current NavStatus message
    navstatus = NavStatus.Data';

    % Ensure NavStatus is good (2) and if not, reset KF
    if navstatus(1) ~= 2
        call(client)
    else
    end

    % Define parameters for navigation algorithm
    K1 = param(3);              % forward velocity gain
    K2 = param(2);              % turning velocity gain
    goaldist = 2;               % distance metric for reaching goal
    goali = 1;                  % current goal index
    xi = param(5);              % attractive force gain
    eta = param(4);             % repulsive force gain
    d = param(1);               % distance above which robot velocity
is constant
    rho0 = param(6);            % offset from obstacle to ignore
repulsive term
    Gx = param(7);              % Terrain gain for force in x
    Gy = param(8);              % Terrain gain for force in y
    thresh = param(9);          % Terrain threshold
    c = 1;                      % initial case variable
    navrun = 0;                 % navigation fix status variable

    % Define parameters for wall-following algorithm
    WallFollow = false;         % initialize wall following flag
    cnt = 0;                    % goal distance counter
    N_Buffer = 20;              % # of measurements for mean LIDAR
force
    RunTurn = 0;                % initialize sum of turns in WF mode
    Flas_Buffer = zeros(1,N_Buffer);  % initialize repulsive force
buffer
```

```
    % Output velocity filter parameters
    Kfilterold = 0.6;           % percentage of old velocity used
    Kfilternew = 0.4;           % percentage of new velocity used
    LinearVel_old = 0.0;        % initialize linear velocity
    AngularVel_old = 0.0;       % initialize angular velocity

    % Set up webcam
    clear cam
    cam =  webcam('Microsoft');

    % Load Terrain Model and FixPts
    load('TerrainModel2.mat', 'TerrainModel')
    load('TerraVars.mat', 'FixPts', 'RanBear', 'XYPos');

    % Load known obstacles, group them, and fit 'lines' to groups
    ObsCheck = exist('knownObs.mat', 'file');
    if ObsCheck == 2
        load('knownObs.mat', 'knownObs')
        [Nbors,~] = NborHood(knownObs, 4);
        RefObs = ObsRevision(Nbors);
    else
        knownObs = [];
    end

    % Initialize values for Terrain Following
    TerrSpot = false;                % Flag for spotted terrain
    TerrConf = false;                % Flag for confirmed terrain
    Fter_Buffer = zeros(1,N_Buffer);  % Initialize Terrain Buffer
    TerFlag = false;                 % Initialize if w/in 0.25m
reached

    % Initialize values for waypoint use
    WayPtNum = 1;
    NumofWayPts = 0;
    waypoints = [];

    % Origin Lat/Lon = [36.583093, -121.881946, 0.0]

    %% State Based Machine
    while 1                     % Infinite loop until goal is reached
        % publish goal coordinates
        goalMsg.Pose.Pose.Position.X = current_goal(2);
        goalMsg.Pose.Pose.Position.Y = current_goal(1);
        goalMsg.Pose.Pose.Orientation.X = 0;
        goalMsg.Pose.Pose.Orientation.Y = 0;
        goalMsg.Pose.Pose.Orientation.Z = 0;
        goalMsg.Pose.Pose.Orientation.W = 1;
        send(goalPub,goalMsg);

        % get the laser ranges
        laser_range = Laser.Ranges;

        % angular resolution vector
```

```
        laser_angle =
(Laser.AngleMin:Laser.AngleIncrement:Laser.AngleMax)';

        % get goal coordinates in XY world frame
        q_goal = [Goal.Pose.Pose.Position.X,
Goal.Pose.Pose.Position.Y];

        % get current GPS fix
        gpsfix = [GPSFix.Status.Service,GPSFix.Status.Status];

        % get current nav status
        navstatus = NavStatus.Data';

        % if good nav status, set nav status variable
        if navstatus(1) == 2
            navrun = 1;
        else
        end

        % if bad nav status with previous good fix and good GPS fix,
reset KF
        if navstatus(1) == 3 && navrun == 1 && gpsfix(2) == 30
            call(client)
            navrun = 0;
        else
        end

         % get X, Y and Theta
        pose = Pose.Pose.Pose;
        quat = pose.Orientation;
        angles = quat2eul([quat.W quat.X quat.Y quat.Z]);
        yaw = angles(1);
        x = pose.Position.X;
        y = pose.Position.Y;
        th = yaw;

        fprintf('X: %f, Y: %f, Theta: %f \n',x,y,th);
        fprintf('quat.W: %f    Yaw: %f\n', quat.W, yaw);

        % call the attractive force function
        wp_x = q_goal(goali,1);
        wp_y = q_goal(goali,2);

        % Find waypoints in the workspace
%        testx = 656; % use these for indoor code check
%        testy = 1345;
%% Waypoints portion
         if isempty(waypoints)
             [waypoints, Status] = AlphaStar([x;y], [wp_x;wp_y],
RefObs);
             WayFlag = true;
             NumofWayPts = size(waypoints,2);
         end

         if WayFlag
```

```
            fprintf('%s \n', Status)
            WayFlag = false;
        end

        if ~isempty(waypoints) && WayPtNum <= NumofWayPts
            gx = waypoints(1,WayPtNum);
            gy = waypoints(2,WayPtNum);
            fprintf('Going to waypoint %f of %f', WayPtNum,
NumofWayPts)
        else
           gx = wp_x;
           gy = wp_y;
           fprintf('Going to goal at %f, %f', wp_x, wp_y)
        end
%% Waypoints portion ends

        [dist, angvel, linvel] = attforcepot(x,y,th,gx,gy,d);

        % evaluate what to do next based on the distance to the
waypoint.
        if (dist <= goaldist)
            % You have reached the goal
            if (WayPtNum <= NumofWayPts)
                % if there are multiple waypoints
                disp('Going to next waypoint!');
                WayPtNum = WayPtNum + 1;
            else
                % Reached the goal
%                 fprintf('WP #%d at x: %f, y: %f, Distance:
%f\n',goalnum,wp_x,wp_y,dist);

                cmdMsg.Linear.X = 0.0;
                cmdMsg.Angular.Z = 0.0;
                fprintf('Publishing cmd_vel with lin. vel: %f, ang.
vel.: %f\n\n', ...
                    0.0,0.0);
                send(cmdPub,cmdMsg);
                disp('Done!')
                break;      % exit while loop as final goal is reached
            end
        else
            % goal not yet reached
            % fprintf('WP #%d at x: %f, y: %f, Distance:
%f\n',goalnum,wp_x,wp_y,dist);

        end
        Fatt = [linvel;angvel];         % from top of loop

        pause(0.1)          % pause for ROS system

        %% LIDAR Portion of Force
        [Flas, LocLas] = LidarForce(laser_range, laser_angle, rho0,
eta);
        MinLaserRange = length(laser_range(laser_range < 0.5));
```

```
        % Check to add location to map
        if norm(xi*Fatt + Flas) < 0.5 && dist > 1 && (MinLaserRange <
3)
            knownObs = knownObsMemory(LocLas,th, x, y, knownObs);
            WallFollow = true;  % flag to enter wall following mode
        end
        Flas_Buffer = [Flas(2), Flas_Buffer(1:end-1)];

        %% Terrain Portion of Force
        Fterr = TerrainForce(cam, TerrainModel, FixPts, RanBear, Gx,
Gy, thresh, eta);
        if Fterr(2) > 0
            yter = -0.5;
        else
            yter = 0.5;
        end
        LocTer = [1;yter];  % Due to constricted field of view and
                                % simplicity this is assumed

        % if terrain is spotted and confirmed we add it to knownObs
        % and enter Terrain Following Mode
        if TerrSpot && norm(Fterr) > 2 && dist > d
            TerrConf = true;   % flag to enter terrain following mode
            knownObs = knownObsMemory(LocTer,th, x, y, knownObs);
        elseif norm(Fterr) > 2 && dist > d
            TerrSpot = true;
        else
            TerrSpot = false;
        end
        Fter_Buffer = [Fterr(2), Fter_Buffer(1:end-1)];

        %% Known Obstacles Portion of Force
        if isempty(RefObs)
            Fknobs = [0;0];
        else
            Fknobs = knownObsForce(x,y,th,RefObs);
        end

        %% switch/case for algorithm decision logic
        switch c
            %% Potential Field Algorithm
            case 1
                fprintf('Potential Field\n')
                caseMsg.Data = 'Potential Field';   % publish current
case to ROS
                send(casePub,caseMsg)

                % calculate total force and build velocity terms
                if TerrConf
                    Ftot = [0;0];
                else
                    Ftot = xi*Fatt + Flas + Fterr + Fknobs;
                end
                fprintf('\n\nFattX: %f\nFattY: %f\n FreptX: %f\nFreptY:
%f\nFterrainX: %f\nFterrainY: %f\nFknobsX: %f\nFknobsY; %f\n',...
```

```
                    xi*Fatt(1),
xi*Fatt(2),Flas(1),Flas(2),Fterr(1),Fterr(2), Fknobs(1), Fknobs(2));
                LinearVel = K1*Ftot(1);
                AngularVel = K2*Ftot(2);

                % determine which case to enter next
                if MinLaserRange > 2
                    cPrior = 1;
                    c = 4;
                elseif WallFollow
                    MeanBuffer = mean(Flas_Buffer);
                    cnt = 0;
                    c = 2;
                elseif TerrConf
                    turn = mean(Fter_Buffer);
                    goalt = TerrainFollow(cam, XYPos, turn, yaw, x, y);
                    c = 3;
                    timer = 0;
                    TerrConf = false;
                    TerrSpot = false;
                else
                    c = 1;
                end
            %% Wall Following Algorithm
            case 2
                fprintf('\nWall Following\n\n')
                caseMsg.Data = 'Wall Following';    % publish current
case to ROS
                send(casePub,caseMsg)

                [LinearVel,AngularVel] = wallFollow(Flas, MeanBuffer);
                cnt = cnt + 1;  % counter
                if cnt > 1
                    RunTurn = RunTurn + AngOut*0.1;
                end % AngOut is the turn rate from the last iteration
                    % 0.1 is approximate time step

                % determine which case to enter next
                if MinLaserRange > 2
                    c = 4;
                    cPrior = 2;
                elseif cnt >= 10  && (RunTurn >= -0.08 && RunTurn <=
0.08)
                    c = 1;
                    WallFollow = false;
                    RunTurn = 0;
                    cnt = 0;
                    Flas_Buffer = zeros(1,N_Buffer);
                    waypoints = [];  % no assumption we can reach the
                                     % waypoints, will replan.
                    % TerrConf = false;
                    % TerrSpot = false;
                else
                    c = 2;
                end
```

```
            %% Terrain Following Algorithm
            case 3
                 fprintf('\nTerrain Following\n\n')
                 caseMsg.Data = 'Terrain Following';    % publish
current case to ROS
                 send(casePub,caseMsg)

                 timer = timer + 1;
                 % Use Potential to temp goal
                 [dist2terr, angvelt, linvelt] =
attforcepot(x,y,th,goalt(1),goalt(2),1);

                 if dist2terr < 1
                     TerFlag = true;
                 end

                 Ftf = [linvelt;angvelt];
                 Ftert = xi*Ftf + Flas + Fterr + Fknobs;
                 LinearVel = K1*Ftert(1);
                 AngularVel = K2*Ftert(2);

                 % determine which case to enter next
                 % emergency avoid
                 if MinLaserRange > 2
                     c = 4;
                     cPrior = 3;
                 % wall follow
                 elseif WallFollow
                     MeanBuffer = mean(Flas_Buffer);
                     cnt = 0;
                     c = 2;
                 % update temp goal, unplanned
                 elseif ~TerFlag && norm(Fterr) > 2
                     knownObs = knownObsMemory(LocTer,th, x, y,
knownObs);
                     goalt = TerrainFollow(cam, XYPos, turn, yaw, x, y);
                     c = 3;
                 % return to potential field
                 elseif TerFlag || (timer > 150  && norm(Fterr) < 1)
                     % reached temp goal or been in the mode for ~30
seconds
                     % and terrain force is not very high
                     c = 1;
                     goalt = [];
                     turn = [];
                     TerFlag = false;
                     TerrConf = false;
                     TerrSpot = false;
                     waypoints = [];
                 % continue in terrain following
                 else
                     c = 3;
                 end
             %% Emergency Avoidance
```

```
            case 4
                ii = 0;
                while ii < 5
                    % stop immediately for 5 seconds
                    fprintf('Emergency Avoidance\n')
                    caseMsg.Data = 'Emergency Avoidance (PF)';
                    send(casePub,caseMsg)
                    % populate the message
                    fprintf('WP #%d at x: %f, y: %f, Distance:
%f\n',goalnum,wp_x,wp_y,dist);
                    fprintf('Yaw: %5.2f\n', quat.W);
                    cmdMsg.Linear.X = 0.0;
                    cmdMsg.Angular.Z = 0.0;
                    % publish message
                    fprintf('Publishing cmd_vel with lin. vel: %f, ang.
vel.: %f\n', ...
                        0.0,0.0);
                    send(cmdPub,cmdMsg);
                    pause(0.2)
                    ii = ii + 0.2;
                end
                jj = 0;
                while jj < 4
                    % backup for 4 seconds to make enough room to
maneuver
                    % around obstacle
                    caseMsg.Data = 'Emergency Avoidance (PF)';
                    send(casePub,caseMsg)
                    fprintf('WP #%d at x: %f, y: %f, Distance:
%f\n',goalnum,wp_x,wp_y,dist);
                    cmdMsg.Linear.X = -0.1;
                    cmdMsg.Angular.Z = 0.0;
                    % publish
                    fprintf('Publishing cmd_vel with lin. vel: %f, ang.
vel.: %f\n', ...
                        0.0,0.0);
                    send(cmdPub,cmdMsg);
                    pause(0.2);
                    jj = jj + 0.2;
                end

                % determine if obstacle is out of minimum range
parameter
                if MinLaserRange > 2
                    c = 4;
                else
                    c = cPrior;
                end
            otherwise
        end

        % build filtered output velocity parameters with bounds
        if Kfilternew*LinearVel + Kfilterold*LinearVel_old > 3
            LinOut = 3;
        elseif Kfilternew*LinearVel + Kfilterold*LinearVel_old < -0.5
```

```
            LinOut = -0.5;
        else
            LinOut = Kfilternew*LinearVel + Kfilterold*LinearVel_old;
        end

        if Kfilternew*AngularVel + Kfilterold*AngularVel_old > pi/3
            AngOut = pi/3;
        elseif Kfilternew*AngularVel + Kfilterold*AngularVel_old < -
pi/3
            AngOut = -pi/3;
        else
            AngOut = Kfilternew*AngularVel + Kfilterold*AngularVel_old;
        end
        cmdMsg.Linear.X = LinOut;
        cmdMsg.Angular.Z = AngOut;

        % publish on cmd_vel topic
        fprintf('Publishing cmd_vel with lin. vel: %f, ang. vel.:
%f\n\n', ...
            cmdMsg.Linear.X,cmdMsg.Angular.Z);
        send(cmdPub,cmdMsg);

        LinearVel_old = cmdMsg.Linear.X;
        AngularVel_old = cmdMsg.Angular.Z;
    end
end
```

APPENDIX E. PATH PLANNING ALGORITHMS

A. A* IMPLEMENTATION

```
%% Approximate Cell Decomposition and A*
% this function takes in the starting points of the robot, the goal,
% and the location of known obstacles; pads it, and outputs a list of
% waypoints path through the cells.
% This function works in the UTM frame or any generic xy defined frame.
% It does not take in or output information in lat/lon

function [waypoints, Status] = AlphaStar(Pose, Goal, knownObs)
    %% Generate the Cells
    cellDim = 1;   % Observationally, the robot shhould be able to
                   % handle a cell of width 1m, without significant
                   % issues
    PoI = [Pose, Goal, knownObs];  % this function assumes that none of
                                   % these reside in the same cell
    % build the padding
    minX = min(PoI(1,:));
    maxX = max(PoI(1,:));
    minY = min(PoI(2,:));
    maxY = max(PoI(2,:));
    bord = 4*cellDim;
    corners = [minX-bord, minX-bord, maxX+bord, maxX+bord;
               minY-bord, maxY+bord, minY-bord, maxY+bord];
    PoI = [PoI, corners];
    [Nbors, MidPoints] = NborHood(PoI, cellDim);

    %% A*

    % find the initial and goal cells
    [R, C] = size(Nbors);
    H = zeros(R,C);    % cost from cell to goal
    ICell = [];        % initialize q_init
    GCell = [];        % initialize q_goal
    for ii = 1:R
        for jj = 1:C
            y = Nbors{ii,jj};
            if ~isempty(y)   % if empty no action
                if max(y(1,:) == Pose(1))
                    if max(y(2,:) == Pose(2))
                        ICell = [ii,jj]; % q_init
                    end
                end
                if max(y(1,:) == Goal(1))
                    if max(y(2,:) == Goal(2))
                        GCell = [ii,jj]; % q_goal
                    end
                end
            end
        end
    end % Cells identified, could be in the same cell
```

```
% empty corners
Nbors{1,1} = [];
Nbors{1,C} = [];
Nbors{R,1} = [];
Nbors{R,C} = [];
if ~isempty(GCell) && ~isempty(ICell)
    % Next we fill H
    for ii = 1:R
        for jj = 1:C
            NN = size(Nbors{ii,jj},2);  % check if cell is occupied
            if NN ~= 0                  % if the cell is not empty
                flag = true;            % track if q_init/q_goal
                                        % were assigned
                if ii == GCell(1) && jj == GCell(2)
                    H(ii, jj) = 0;      % h to q_goal is 0
                    flag = false;
                end
                if ii == ICell(1) && jj == ICell(2)
                    H(ii,jj) = norm(MidPoints{ii,jj} -...
                    Goal); % h to q_init is
                                        % 2-norm to q_goal
                    flag = false;
                end
                if flag
                    H(ii,jj) = inf;
                end
            else                        % internal map indicates
                                        % empty cell
                H(ii,jj) = norm(MidPoints{ii,jj} -...
                    MidPoints{GCell(1), GCell(2)});
            end
        end
    end

% this check method places infinite cost in cells only containing
% obstacles, any that have obstacles and goal or init will be
% assigned a cost and online obstacle avoidance will have to handle
% it.

% now for some image processing techniques
ObsOutline = isinf(H);
nhood = ones(2);
se = strel(nhood);
ObsSpace = imdilate(ObsOutline,se);  % this grows every obstacle
                                     % using the morphological 3x3
                                     % neighborhood, it gives
                                     % standoff and closes gaps in
                                     % occluded cells so that we
                                     % don't plan a path between
                                     % them (based on assumption
                                     % that obstacles are
                                     % continuous)
ObsSpace = bwmorph(ObsSpace, 'skel', inf);
ObsSpace = bwmorph(ObsSpace, 'diag');
```

```
    H(ObsSpace) = inf;
    % ensure q_init and q_goal are not set to inf
    H(GCell(1), GCell(2)) = 0;
    H(ICell(1), ICell(2)) = norm(MidPoints{ICell(1),ICell(2)} -...
                        MidPoints{GCell(1), GCell(2)});

    % Now we build the path to the goal
    goalReached = false;
    N = ICell;
    CostMat = zeros(R,C);
    visited = zeros(R,C);
    Path = cell(R, C);
    CostMat(N(1),N(2)) = H(N(1),N(2));
    Path{N(1), N(2)} = ICell;
    Status = 'No Info';
    StatFlag = false;
    waypoints = [];
    while ~goalReached
        for ii = -1:1
            if N(1) == 1 && ii == -1              % Boundary check row
            elseif N(1) == R && ii == 1
            else
                for jj = -1:1
                    if N(2) == 1 && jj == -1      % Boundary check cols
                    elseif N(2) == C && jj == 1
                    elseif ~((jj == 0) && (ii == 0)) % do not check 0,0
                        runGoCost = CostMat(N(1),N(2))-H(N(1),N(2));
                        % the go cost to the origin cell along the
                        % shortest path found thus far
                        goCost = norm([ii;jj])*3; % cost to get to the
                        % cell being inspected from the origin cell
                        % (not q_init)
                        HCost = H(N(1) + ii, N(2) + jj);
                        % 2-norm from midpoint of cell to midpoint
                        % of goal cell (inf if obstacle present)
                        costTemp = runGoCost + goCost + HCost;
                        % potential cost to get to the cell in question

                        % assign cost from first encounter
                        if CostMat(N(1) + ii, N(2) + jj) == 0
                            CostMat(N(1) + ii, N(2) + jj) = costTemp;
                            Path{N(1) + ii, N(2) + jj} = [Path{N(1),
N(2)}; [N(1) + ii, N(2) + jj]];
                        % update if cost is cheaper
                        elseif CostMat(N(1) + ii, N(2) + jj) > costTemp
                            CostMat(N(1) + ii, N(2) + jj) = costTemp;
                            Path{N(1) + ii, N(2) + jj} = [Path{N(1),
N(2)}; N(1) + ii, N(2) + jj];
                        end  % if otherwise make no changes
                    end
                end
            end
        end
        visited(N(1),N(2)) = 1;   % indicates the origin cell has been
                                  % visited
```

```
        %% determine the next cell to check
        % lowest cost, and unvisited
        ind = find(CostMat > 0);  % provides indices of nonzero values
                                  % in cost matrix
        [~,I] = sort(CostMat(ind), 'ascend'); % provides index of the
                                              % sorted costs in
                                              % ascending order
        ind = ind(I);             % orders the indices from the lowest
                                  % cost to the highest

        n = 1;
        Next = false;
        while ~Next               % checks the visited matrix, if one
                                  % checks again.  This increments the
                                  % indices to check the next lowest
                                  % cost, always starts at one

            [row, col] = ind2sub([R,C], ind(n));
            if visited(row,col) == 0
                Next = true;
                N = [row, col];
            else
                n = n + 1;
            end
            if n > length(ind)  % this doesn't seem possible, but it is
                StatFlag = true;
                break;
            end
        end

        if N(1) == GCell(1) && N(2) == GCell(2)
            goalReached = true;
            for ll = 1:length(Path{N(1),N(2)})
                waypoints = [waypoints,
MidPoints{Path{N(1),N(2)}(ll,1),...
                                              Path{N(1),N(2)}(ll,2)}];
                Status = 'Path found!';

            end
            way1 = downsample(waypoints(1,:),4,1);
            way2 = downsample(waypoints(2,:),4,1);
            waypoints = [way1;way2];
                                  % we are already in the initial
                                  % cell, and the downsampled
                                  % version smooths the overall path
        elseif StatFlag
            waypoints = Goal;
            Status = 'Heading to Goal!';
        end
    end
    else
    waypoints = Goal;
    Status = 'Heading to Goal!';
    end
end
```

B. APPROXIMATE CELL DECOMPOSITION

```
%% NborHood
% this script is designed to divide the area that has known obstacles
% into cells that will then be fed to a separate function to refine the
% placement of those obstacles in order to alleviate jagged edges when
% dealing with the control algorithm, and for path planning.

function [Nbors, MidPoints] = NborHood(knownObs, Inc)
    cols = ceil((max(knownObs(1,:))-min(knownObs(1,:)))/Inc);
    rows = ceil((max(knownObs(2,:))-min(knownObs(2,:)))/Inc);
    startX = min(knownObs(1,:));
    startY = min(knownObs(2,:));

    Nb = cell(rows, cols);  % initialize Neighborhood cell array
    MP = cell(rows, cols);  % initialize Midpoint cell array

    for ii = 1:rows         % Populate neighborhoods and midpoints
        for jj = 1:cols
            Lrow = double(knownObs(1,:) >= startX + Inc*(jj-1));
            Urow = double(knownObs(1,:) < startX + Inc*(jj));
            BoundRow  = (Lrow==Urow);
            subRow = knownObs(:,BoundRow);
            Lcol = double(subRow(2,:) >= startY + Inc*(ii-1));
            Ucol = double(subRow(2,:) < startY + Inc*(ii));
            BoundCol  = (Lcol==Ucol);
            if max(BoundCol) == 1 % something is in the neighborhood
                Nb{ii,jj} = subRow(:,BoundCol);
            else                  % neighborhood is empty
                Nb{ii,jj} = [];
            end
            MP{ii,jj} = [startX + Inc*(jj-2) + 0.5*Inc;
                         startY + Inc*(ii-2) + 0.5*Inc];
        end
    end
    Nbors = Nb;
    MidPoints = MP;
end
```

C. OBSTACLE REVISION

```
%% ObsRevision
% revision of the placement of known obstacles using least mean
% squares, with linear assumption.  This function is fed by the
% NborHood function to find the points that are in the same
% neighborhood

function NewObs = ObsRevision(Nbors)
    [R,C] = size(Nbors);   % determines range for the double 'for' loop
    NewObs = [];           % initializes our output

    for ii = 1:C
        for jj = 1:R
            y = Nbors{jj,ii};
```

```
            len = length(y);  % value is used throughout
            newObs = [];   % initializes the output from each cell
            if len == 0
            elseif len > 2 % computes the LMS solution for line to the
                           % points in the neighborhood
                c1 = ones(len,1); % constant in linear regression
                c2 = (0:len-1).'; % slope in linear regression
                M = [c1,c2];      % Matrix used to find the line that
                                  % best fits the neighborhood
                % the following is done bc order matters in LMS
                [minX, maxX] = bounds(y(1,:));
                [minY, maxY] = bounds(y(2,:));
                dx = maxX - minX;
                dy = maxY - minY;
                if dx > dy
                    [~,I] = sort(y(1,:),'ascend');
                else
                    [~,I] = sort(y(2,:),'ascend');
                end
                y = y(:,I);
                Beta = (M.'*M)^(-1)*M.'*y.'; % this is the constant and
                                             % slope associated to x
                                             % and y
                cnt = 1;   % iteration variable due to bounding
                for ll = 1:len*2 % assuming we know there is a feature
                                 % in this cell we can improve control
                                 % in that region by adding other
                                 % points
                    temp = Beta(1,:).' + Beta(2,:).'*(ll-1);
                    if dx > dy
                        if temp(1) >= y(1,1) && temp(1) <= y(1,end)
                            newObs(:,cnt) = temp;
                            cnt = cnt+1;
                        end
                    else
                        if temp(2) >= y(2,1) && temp(2) <= y(2,end)
                            newObs(:,cnt) = temp;
                            cnt = cnt+1;
                        end
                    end
                end
            elseif sum(y == [0;0]) ~= 2
                newObs = y;
            end
            NewObs = [NewObs, newObs];
        end
    end
end
```

APPENDIX F. FORCES

A. ATTRACTIVE FORCE

```
%% Attractive Force
% Function adapted from Calvin Hargadine's thesis code to split up
% code and save space.

function [dist,angvel,linvel] = attforcepot(x,y,th,wp_x,wp_y,d)
    maxvel = 3;

    dist = sqrt((wp_x-x)^2+(wp_y-y)^2);
    ang = atan2((wp_y-y),(wp_x-x));
    angerror = ang-th;

    while angerror > pi
        angerror = angerror-2*pi;
    end
    while angerror < -pi
        angerror = angerror+2*pi;
    end

    angvel = angerror;

    if dist <= d
        linvel = dist;
    else
        linvel = maxvel;
    end
end
```

B. REPULSIVE FORCES

1. Force Due to LIDAR

```
%% LIDAR Force
% Function modifies original repulsive force due to LIDAR readings

function [Flaser, LocFlaser] = LidarForce(laser_range, laser_angle,
rho0, eta)
     Flaser = [0;0];      % initialize repulsive force

    for i = 1:1032
        if laser_range(i) <= 20
            % object position in the laser i coordinate in meters
            p_laser = [laser_range(i) 0 0 1]';
            Xobj = cos(laser_angle(i))*p_laser(1);
            Yobj = sin(laser_angle(i))*p_laser(1);
            rho = sqrt(Xobj^2+Yobj^2);
            if rho < rho0
                Frep = eta*(1/p_laser(1)-1/rho0)*(1/(p_laser(1)^2))*[-
cos(laser_angle(i)) -sin(laser_angle(i))]';
            else
```

```
                Frep = [0;0];
            end
            Flaser = Flaser+Frep;
        else
        end
    end

    rmean = mean(laser_range(laser_range < rho0));
    mangle = mean(laser_angle(laser_range < rho0));
    LocFlaser = [rmean*cos(mangle); rmean*sin(mangle)];
end
```

2. Force Due to Terrain

```
%% Terrain Force
% function develops force from hazardous terrain

function Fterrain = TerrainForce(cam, model, fxPts, RanBear, Gx, Gy,
thresh, eta)
    %% Grab Photo
    I = snapshot(cam);
    I = rgb2gray(I);

    H = 40;
    W = 64;

    %% Get Features
    [features, ~] = extractFeatures(I, fxPts,...
    'Method', 'Block', 'BlockSize', 11); % lifts the pixel values in an
                                         % 11x11 cell and normalizes
                                         % to build feature vector

    features = im2double(features);
    [label, ~] = predict(model, features);% currently not using score

    PredVec = str2double(label);        % TreeBagger 'label' is a cell
                                        % with char elements

    h = (1/9)*ones(3,3);                % 3x3 Box Filter

    PredMat = reshape(PredVec, H, W);   % Reshapes Vec to Mat for
                                        % geometric sense in filtering

    PredMat = conv2(PredMat, h,'same'); % smooths PredMat
    PredMat = round(PredMat);           % thresholds PredMat

    PredMat = bwareaopen(PredMat, 128); % any connected regions w/ less
                                        % than 128 elements are removed

    JJ = PredMat > thresh;              % logical of values greater
than
                                        % the threshold
    Temp = cell2mat(RanBear(JJ));       % matrix reshaped by column,
                                        % ENU frame
```

```
    %% Build Terrain Force Vector
    if isempty(Temp)                    % If clear terrain, no
repulsive force
        Fterrain = [0;0];
    else                                % Build force to avoid detected
terrain
        Fterrain = [0;0];
        X = size(Temp,2);
        for ii = 1:X
            Fterra = -eta*(1/Temp(1,ii)-1/2.5)*...
                     [Gx*cos(Temp(2,ii)); Gy*sin(Temp(2,ii))];
            Fterrain = Fterrain + Fterra;
        end
    end
end
```

3. Force Due to Known Obstacles

```
%% Known Obstacle Force
% develops force from known obstacles within region of influence

function [Fknobs] = knownObsForce(x,y,th,knownObs)
    Rho0 = 6;
    Eta0 = 2;
    Fknobs = [0;0];   % initialize known obstacle force
    dist2obs = sum(([x;y]-knownObs).^2).^(1/2); % finds the distance
                                                % to all obstacles
    [dist2obs, I] = sort(dist2obs,'ascend'); % orders the distance from
                                             % closest to farthest
    KnownTemp = knownObs(:,I);        % the positions of the obstacles
                                      % sorted from closest to farthest
    for ii = 1:3      % contribution from 3 closest obstacles
        if dist2obs(ii) < Rho0
            % obstacle postion in robot frame
            XYr = [cos(th) -sin(th);sin(th) cos(th)].'*...
                ([KnownTemp(1,ii);KnownTemp(2,ii)]-[x;y]);
            Ftemp = -Eta0*(1/dist2obs(ii)-1/Rho0)*...
                (1/dist2obs(ii)^2)*XYr;
        else
            Ftemp = [0;0];
        end
        Fknobs = Fknobs + Ftemp;
    end   % When this loop is complete all known obstacles will have
          % been checked and their contribution included
    if Fknobs(1) < -3
        Fknobs(1) = -3;  % limits the extent of the repulsion
    end
end
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX G. ESCAPE MODES

A. WALL FOLLOWING

```
%% Wall Following Function
% determines the velocities while in wall following mode
% implements Calvin Hargadine's method as a function

function [linvel, angvel] = wallFollow(Flas,MeanBuffer)
    angK = 1;                % turning velocity gain for WF algorithm
    linK = 1;                % forward velocity gain for WF algorithm

    % determine angle to the repulsive force vector
    objang = atan2(Flas(2),Flas(1));
    if objang < 0
        objang = objang + 2*pi;
    else
    end

    objangdeg = objang*180/pi;

    % determine which way to turn and keep repulsive force vector
    % perpendicular with robot heading
    if MeanBuffer > 0
        if objangdeg >= 100
            angvel = angK*0.4;
            linvel = linK*0.05;
        elseif objangdeg < 80
            angvel = -angK*0.4;
            linvel = linK*0.05;
        else
            angvel = 0.0;
            linvel = 0.3;
        end
    elseif MeanBuffer < 0
        if objangdeg < 260
            angvel = -angK*0.4;
            linvel = linK*0.05;
        elseif objangdeg > 280
            angvel = angK*0.4;
            linvel = linK*0.05;
        else
            angvel = 0.0;
            linvel = 0.3;
        end
    end
end
```

B. TERRAIN FOLLOWING

```
%% Terrain Following Function
% this function outputs a temporary goal for the robot to follow the
% terrain.
```

```
function goalt = TerrainFollow(cam, XYPos, turn, yaw, x, y)
    I = snapshot(cam);
    I = rgb2gray(I);

    %% Need ang2goal

    %% Terrain Orientation
    BW = edge(I, 'sobel');
    [HH, TT, RR ] = hough(BW);
    peaks = houghpeaks(HH, 10,'threshold', ceil(0.7*max(HH(:))));
    lines = houghlines(BW, TT, RR, peaks);

    % turn the points in the photo to points in the reference frame
    ThetaTerr = zeros(1,length(lines));
    for jj = 1:length(lines)
        xy1 = lines(jj).point1;
        xy2 = lines(jj).point2;
        XY1 = XYPos{xy1(2), xy1(1)};
        XY2 = XYPos{xy2(2), xy2(1)};
        ThetaTerr(jj) = atan2(XY2(2)-XY1(2), XY2(1)-XY1(1));
    end

    % Find the average angle of the lines found
    ThetaMean = mean(ThetaTerr);
    ThetaTerr = ThetaTerr(ThetaTerr > ThetaMean - pi/18);
    ThetaTerr = ThetaTerr(ThetaTerr < ThetaMean + pi/18);
    if ~isempty(ThetaTerr)
        Theta = mean(ThetaTerr);
    else
        Theta = ThetaMean;
    end

    % constrain  -pi <= theta <= pi
    if Theta > pi/2
       Theta = Theta-pi;
    elseif Theta < -pi/2
        Theta = Theta+pi;
    end

    % ensure the temp goal is placed on the appropriate side
    if turn > 0 && Theta < 0
        Theta = Theta + pi;
    elseif turn < 0 && Theta > 0
        Theta = Theta -pi;
    end

    % translate point in robot frame to world frame
    Pb = 2.5*[cos(Theta);sin(Theta)];
    R = [cos(yaw) -sin(yaw);
         sin(yaw) cos(yaw)];
    Pa = R*Pb + [x;y];
    goalt = Pa;
end
```

LIST OF REFERENCES

[1] Department of Defense, “Summary of the 2018 Department of Defense artificial intelligence strategy,” Washington, DC, USA, 2019.

[2] Marine Corps Warfighting Laboratory/Futures Directorate, “2018 U.S. Marine Corps S&T strategic plan,” Quantico, VA, USA, 2018.

[3] S. J. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*. 3rd ed. Upper Saddle River, NJ, USA: Prentice Hall Press, 2009.

[4] C. S. Hargadine, “Mobile robot navigation and obstacle avoidance in unstructured outdoor environments,” M. S. thesis, Dept. of Elec. and Comp. Eng., NPS, Monterey, CA, USA, 2017. [Online]. Available: https://calhoun.nps.edu/handle/10945/56937

[5] M. R. Audette, “Interactive map making for route planning and obstacle avoidance in an unstructured outdoor environment,” M. S. thesis, Dept. of Elec. and Comp. Eng., NPS, Monterey, CA, USA, 2018. [Online]. Available: https://calhoun.nps.edu/handle/10945/60406

[6] P. Papadakis, “Terrain traversability analysis methods for unmanned ground vehicles: A survey,” *Eng. Appl. of Artificial Intell.*, vol. 26, no. 4, pp. 1373–1385, Apr. 2013. [Online]. doi: 10.1016/j.engappai.2013.01.006

[7] P. I. Corke, *Robotics, Vision and Control: Fundamental Algorithms in MATLAB*. Berlin, Germany: Springer, 2011. [Online]. doi: 10.1007/978-3-642-20144-8

[8] Y. N. Khan, P. Komma, and A. Zell, “High resolution visual terrain classification for outdoor robots,” *2011 IEEE Int. Conf. Comput. Vision Workshops (ICCV Workshops)*, Barcelona, Spain, 2011, pp. 1014–1021. [Online]. doi: 10.1109/ICCVW.2011.6130362

[9] J. Latombe, *Robot Motion Planning*, 2nd ed. Boston, MA, USA: Kluwer, 2003.

[10] Adept Technology, Inc., *Pioneer 3-AT*, 2011. [Online]. Available: https://www.generationrobots.com/media/Pioneer3AT-P3AT-RevA-datasheet.pdf

[11] *Pioneer 3 Operations Manual*, MobileRobots Inc., Amherst, NH, USA: MobileRobots Inc., 2006. [Online]. Available: http://vigir.missouri.edu/~gdesouza/Research/MobileRobotics/Software/P3OpMan5.pdf

[12] CappucinoPC, “SlimPRO SP675P mini PC,” Accessed July 18, 2019. [Online]. Available: http://www.cappuccinopc.com/slimpro-sp675p.asp

[13] Hokuyo Automatic Co., “UTM-30LX,” July 18, 2019. [Online]. Available: https://www.hokuyo-aut.jp/search/single.php?serial=169

[14] LORD MicroStrain, *3DM-GX5-45 datasheet*, 2018. [Online]. Available: https://www.microstrain.com/sites/default/files/3dm-gx5-45_datasheet_8400-0091_0.pdf

[15] *3DM-GX5-45 GNSS-Aided Inertial Navigation System*, LORD MicroStrain, Williston, VT, USA: LORD MicroStrain, 2017. [Online]. Available: https://www.microstrain.com/sites/default/files/3dm-gx5-45_user_manual_8500-0010_0.pdf

[16] Microsoft Corporation, *Microsoft (R) LifeCam HD-3000*, 2016. [Online]. Available: https://dl2jx7zfbtwvr.cloudfront.net/specsheets/WEBC1010.pdf

[17] MathWorks, “What is MATLAB,” Accessed July 18, 2019. [Online]. Available: https://www.mathworks.com/discovery/what-is-matlab.html

[18] MathWorks, “Computer vision toolbox,” Accessed July 18, 2019. [Online]. Available: https://www.mathworks.com/products/computer-vision.html

[19] MathWorks, “Image processing toolbox,” Accessed July 18, 2019. [Online]. Available: https://www.mathworks.com/products/image.html

[20] MathWorks, “Mapping toolbox,” Accessed July 18, 2019. [Online]. Available: https://www.mathworks.com/products/mapping.html

[21] MathWorks, “Statistics and machine learning toolbox,” Accessed July 18, 2019. [Online]. Available: https://www.mathworks.com/products/statistics.html

[22] MathWorks, “Webcam support from MATLAB,” Accessed July 18, 2019. [Online]. Available: https://www.mathworks.com/hardware-support/matlab-webcam.html

[23] Open Source Robotics Foundation, “About ROS,” Accessed July 18, 2019. [Online]. Available: https://www.ros.org/about-ros/

[24] Open Source Robotics Foundation, “Core components,” Accessed July 18, 2019. [Online]. Available: https://www.ros.org/core-components/

[25] Google, “Keyhole markup language,” Accessed July 18, 2019. [Online]. Available: https://developers.google.com/kml/

[26] P. E. Hart, N. J. Nilsson, and B. Raphael, “A formal basis for the heuristic determination of minimum cost paths,” *IEEE Trans. Syst. Sci. Cyber.*, vol. 4, no. 2, pp. 100–107, 1968. [Online]. doi: 10.1109/TSSC.1968.300136

[27] L. Breiman, “Random forests,” *Mach. Learn.*, vol. 45, no. 1, pp. 5–32, 2001. [Online]. doi: 10.1023/A:1010933404324

[28] K. P. Murphy, *Machine Learning: A Probabilistic Perspective*. Cambridge, MA, USA: The MIT Press, 2012.

[29] L. Rokach and O. Maimon, “Top-down induction of decision trees classifiers—a survey,” *IEEE Trans. Syst., Man Cybern. C Appl. Rev.*, vol. 35, no. 4, pp. 476–487, Nov. 2005. [Online]. doi: 10.1109/TSMCC.2004.843247

[30] MathWorks, “Classification,” Accessed July 18, 2019. [Online]. Available: https://www.mathworks.com/help/stats/examples/classification.html#d117e4415

[31] P. Filitchkin and K. Byl, “Feature-based terrain classification for LittleDog,” *2012 IEEE/RSJ Int. Conf. Intell. Robots and Syst.*, Vilamoura-Algarve, Portugal, 2012, pp. 1387–1392. [Online]. doi: 10.1109/IROS.2012.6386042

[32] MathWorks, “TreeBagger,” Accessed July 18, 2019. [Online]. Available: https://www.mathworks.com/help/stats/treebagger.html

[33] *Statistics and Machine Learning ToolboxTM User’s Guide*, ver. 11.5, The MathWorks, Inc., Natick, MA: The MathWorks, Inc., 2019. Accessed February 11, 2019. [Online]. Available: https://www.mathworks.com/help/pdf_doc/stats/stats.pdf

[34] Y. Koren and J. Borenstein, “Potential field methods and their inherent limitations for mobile robot navigation,” in *Proc. 1991 IEEE Int. Conf. on Robot. and Autom.*, Sacramento, CA, USA, 1991, pp. 1398–1404. [Online]. doi: 10.1109/ROBOT.1991.131810

[35] X. Yun and K. Tan, “A wall-following method for escaping local minima in potential field based motion planning,” in *1997 8th Int. Conf. Adv. Robot. Proc. ICAR’97*, Monterey, CA, USA, 1997, pp. 421–426. [Online]. doi: 10.1109/ICAR.1997.620216

[36] C. Murphy and H. Singh, “Rectilinear coordinate frames for deep sea navigation,” in *2010 IEEE/OES Auton. Underwater Veh.*, Monterey, CA, USA, 2010, pp. 1–10. [Online]. doi: 10.1109/AUV.2010.5779654

[37] Google Earth Pro 7.3.2.5776 (64-bit). (June 17, 2017). “Naval Postgraduate School.” 36.595° N, 121.876° W, Eye alt 659 ft. Accessed July 18, 2019.

THIS PAGE INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California